SFA Modernization Partner

United States Department of Education

Student Financial Assistance

Integrated Technical Architecture
Detailed Design Document

Volume 1 – Conceptual Architecture

Task Order #16

Deliverable # 16.1.2

October 13, 2000

Table of Contents

List of Figures

List of Tables

1 Preface

1.1. About This Document

The purpose of this document is to provide the architectural foundation on which future applications at Department of Education (DOE) are to be developed. This document is intended to provide Department of Education's application architects with a blue print for developing systems which meet the company's long term computing goals. It is not the purpose of this document to provide an exhaustive inventory of Student Financial Assistance (SFA) existing application portfolio, nor is the intent of this document to address in a complete manner all aspects of the application development life cycle. Each element of the architecture described herein requires logical predecessor / follow-on activities to ensure their realisation, adoption and risk management.

1.2. Intended Audience

This document has been developed to provide Information Technology (IT) managers and architects with a conceptual understanding of Department of Education's Enterprise Technical Architecture (ETA). This document could also be used to train new development staff about Department of Education's architectural goals and conceptual computing topologies.

1.3. Related Documentation

- Department of Education Task Order 4

1.4. Architectural Assumptions

The following list describes those assumptions on which the technical architecture was based.

- Department of Education is moving toward a web centric self-service computing environment.
- Department of Education's goal is to implement an autonomous Enterprise Data Model supported by a data warehouse.
- The standard transport protocol between client and servers will be Transmission Control Protocol / Internet Protocol (TCP/IP) with in Department of Education's distributed architecture. Systems Network Architecture (SNA) will be used within the OS/390 environment.

1.5. Issues

The following issues were unresolved when this version of the technical architecture document was completed:

- None

1.6. TBD Items

The following items affect the implementation of this design and have yet to be defined and will be included in a subsequent release of this design:

- None

2 Introduction

2.1. Business Imperatives

The following business imperatives resulted in the necessity for the Department of Education SFA to move to an alternative technical architecture for enterprise computing.

- Technological changes that resulted in competitive pressures specifically the Internet.
- An aging IT infrastructure that is not capable of keeping pace with changing business requirements.
- SFA needed to more closely align their IT model with the business model.

By mirroring the functional layers of the Business Model the IT Model can be closely aligned and a successful technical architecture can be designed. The correlation between the layers of the Business Model and the IT Model is depicted in Figure 1.

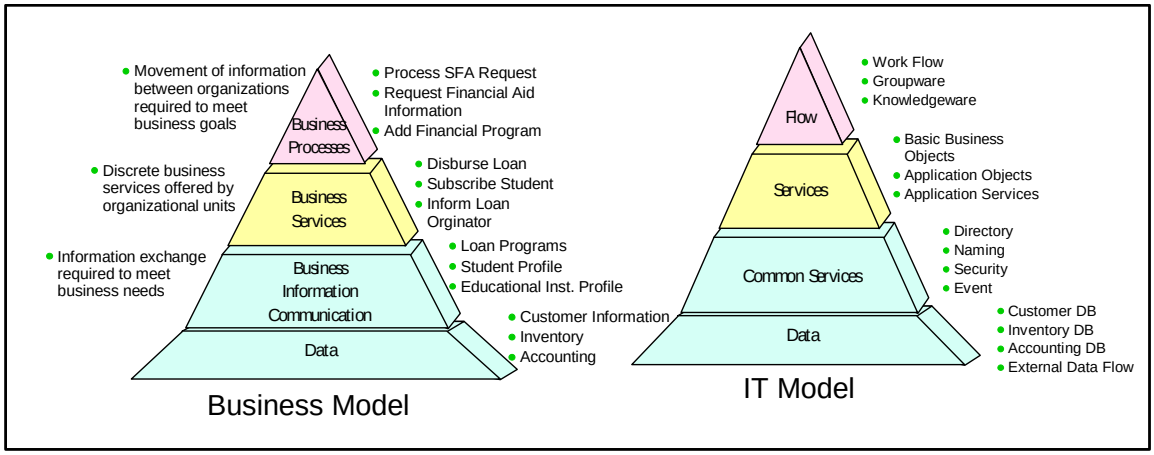

Figure 1 - Correlation between Business and IT Model Layers

2.2. Integrated Technical Architecture Strategy

The Department of Education's Integrated Technical Architecture (ITA) is driven by a need to create an environment that facilitates the integration of disparate data resources, business services and process automation thereby eliminating the barriers created by the following:

- Heterogeneous computing architectures
- Heterogeneous hardware platforms
- Heterogeneous operating systems
- Heterogeneous network configurations, topologies and protocols
- Heterogeneous representations of the same data within the enterprise

In order to meet these goals the architectural implementation must be flexible, easily modified and provide for reuse as the following changes to the enterprise computing environment are introduced:

- A shift in business processes
- Additional data resources
- Future technology shifts
- A move to provide self-service access to SFA

2.3. Design Points

The following requirements and qualifying factors influenced the design of Department of Education's SFA Integrated Technical Architecture.

2.3.1. Legacy Reuse

The Department of Education's current architecture consists of well-established legacy applications and services that are based on a centralized 2 tiered architecture. The future architecture must be able to leverage and extend this investment while enabling a flexible approach to implementing the proposed architecture. While a move to implement the proposed architecture in a short period of time is desirable, reality dictates that the current computing infrastructure will remain until the proposed architecture can be designed, prototyped and successfully implemented. Indeed some elements of the current infrastructure may remain after implementing an interface to extend services into the proposed architecture.

In order to facilitate business process modeling and data independence, additional data repositories will be implemented over time. The additional repositories will include Data Warehouses, Data Marts and Operational Data Stores (ODS). For simplicity these new data stores will be referred to as the Enterprise Data Domain (EDD). The EDD will support a more normalized enterprise data schema than the Legacy Data Domain (LDD). Once implemented, and supported by the proper services the official instances of data will be those stored in the EDD. While current infrastructure components may remain until deprecated, data integrity must be maintained between stores for all data duplicated in both repositories (LDD and EDD).

2.3.2. Distributed Component Architecture

A Component is a piece of software that extends a known interface and provides a set of services. Services can only be provided to the client if the component interface requirements are properly satisfied. A distributed component is a service or application that is based on one or more components that are geographically distributed. The term geographically distributed refers to the architectural topology not necessarily the physical location. Distributed components could reside on the same system but reside in different regions. Distributed components must have the ability to intercommunicate in a physically and geographically distributed environment.

Distributed Components allow processing to be shared among multiple computers (or logical segments) with each computer in the network handling that portion of the overall work for which it is best suited. Distributed Components provide an excellent boundary and segmentation for implementing WorkLoad Management (WLM). Distributed Components will allow Department of Education to scale services using WLM and Load Balancing.

2.3.3. Layered Architecture

A layered architecture helps to provide structure to applications and services that can be decomposed into groups of subtasks in which each group of subtasks is at a particular level of abstraction. An example of a layered architecture is depicted in Figure 2. A layered architecture based on ever increasing abstractions of the previous layers has the following characteristics:

- Increased reusability between layers
- Decrease in ripple effect changes
- Easily modified
- Support for standardization
- Exchangeability with other implementations

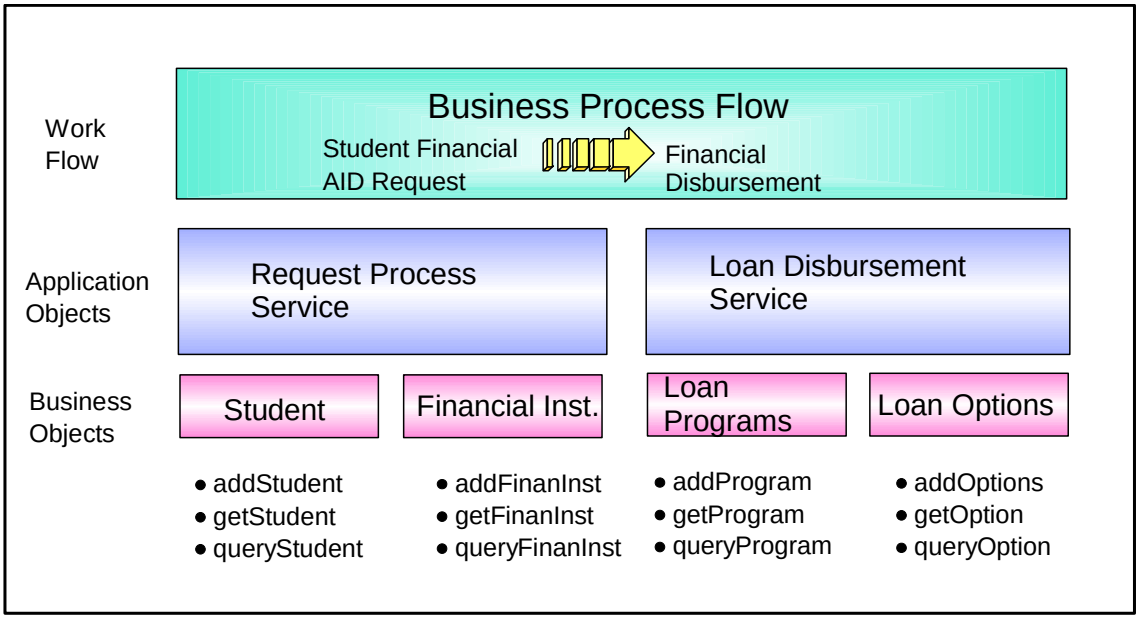

Figure 2 - Layered Component Architecture

2.3.4. Platform Independence

With the advent of client-server computing and the Internet came the need to support different hardware and OS platforms when considering a technical architecture to support

the enterprise. The predominant platforms currently used by Department of Education are as follows:

- OS 390
- VAX / VMS
- UNIX – Solaris, HP / UX
- MS-Windows, NT / 2000 / 95 / 98

The target architecture must be able to support a single development and programming environment. Applications must be deployable on each platform with a minimum of change (if any).

2.3.5. Core Support for Open/Industry Standards

Basing the future technical architecture on Open Industry Standards increases Department of Education's ability to utilize and interact with third party products. In Addition, utilizing Open Industry Standards keeps The Department of Education from being locked into any one vendor's solution. As part of Task Order (TO) 4, the SFA Partners selected a set of products that support DOE requirements and support open industry standards. This document defines the technical architecture mandated by the products selected in Task Order 4. A list of the standard products may be found in the TO-4 deliverables.

2.3.6. Separation of Responsibilities

The distributed component model drives decoupling the client interface framework from the business logic and data. This decoupling helps to provide a separation of responsibilities between the development staff. This allows client developers to concentrate on client frameworks and presentation logic while business logic developers work on application components. The following is a list of possible staff positions resulting from the implementation of this architecture:

- The **Client Developer** is responsible for developing the user interface and the client frameworks that provides a higher level of abstraction for accessing objects and services using the underlying common services. Typically Client Developers will also develop application client frameworks based on the Presentation-Abstraction-Control (PAC) pattern. The PAC pattern provides an abstraction that separates the human-computer interaction from the information-processing components thereby increasing modularity and reuse.
- The **Business Object Developer** is responsible for implementing the business logic using a particular component model. For example, Enterprise Java Beans (EJB) or Common Object Request Broker Architecture (CORBA). These components include basic business components, business application components and business integration components. This responsibility includes the process of wrapping back end data stores using object relational mapping techniques. Often this responsibility is separated into another development role called the **Data Object Developer**.

- The **Data Object Developer** or **Deployment Developer** is responsible for defining the semantic mapping and construction of wrapping back end resource managers [transactional and relational data stores] using object mapping techniques.
- The **Common Services Developer** is responsible for administrating and coordinating middleware services with the implementation of business applications. These middleware services are often referred to as common services. They provide crucial services to the applications such as directory services (Lightweight Directory Access Protocol (LDAP)) or security (Kerberos). In some cases this developer must help implement frameworks to provide access to common services that require unsupported or enterprise specific application interfaces.
- The traditional **Database Developer** is responsible for implementing enterprise data schemas and maintaining both the data and performance integrity of the databases. This is still true in a component-based development environment with the exception of some additional responsibilities. These additional responsibilities include working with the Business Object Developer to establish a schema that facilitates an object relational mapping of business components. Often object relational mapping requires some normalization tradeoffs that the Database Developer must realize and support. The Database developer should have some training in the development of business components.

2.4. A Component-Based Technical Architecture

Department of Education's future Technical Architecture is based on using components as building blocks to develop applications and services. EJB will be the primary standard component model used within SFA. The underlying services used by the Enterprise Java Server (EJS) and all EJBs will be supplied by CORBA services. In some cases SFA developers may supplement the EJB components with CORBA Business Objects. The technique of using components as building blocks to implement services is often referred to as a 'component framework' or in the Object-Oriented (OO) world an 'application framework.' Essentially, a component framework extends a set of interfaces and provides a well known set of services which are dictated by roles of interaction that govern how components 'plugged into' the framework may interact (Figure 3). The components that arise from this definition have the following characteristics:

- Are independent units of deployment
- Are units of third-party composition
- Has no persistent state as a whole

While the use of an object oriented programming model is not required in order to implement a component, Department of Education will standardize on the use of an object oriented programming language to implement components. Where possible the standard object oriented programming language will be Java. However, it is acceptable to use C++ in cases where only C / C++ programming Application Program Interfaces (API) are available.

90% of all I/S shops will adopt compnent-based development models within two years.
By 2001, 75% of all new applications will be deployed via component technology. *Source: Gartner Group*

Components are the fastest growing segment in the application development market (85% CAGR). *Source: IDC*

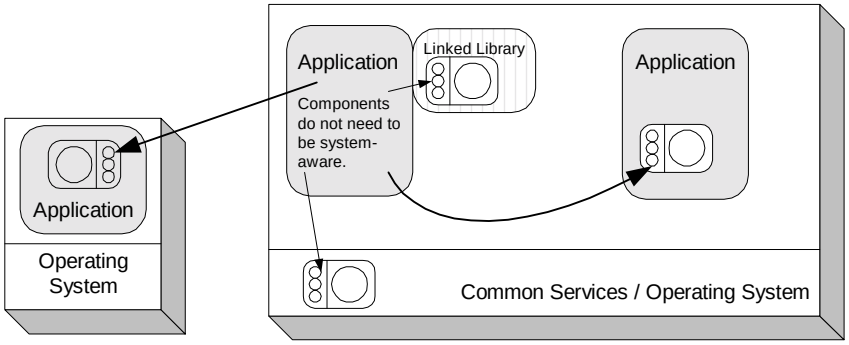

Figure 3 - Component Architecture

3 Selecting Standard Component Architectural Patterns

Viable software systems are developed according to some overall structuring principle. These principles are described using an *architectural pattern*. An architectural pattern establishes a fundamental structural organization on which to base a software implementation. It provides a framework in which a particular architectural problem can be solved. Architectural patterns are templates for technical architectures. The selection of an architectural pattern determines the strategy for developing a software system.

The following sections define the architectural patterns that will be used in the implementation of the SFA technical architecture.

3.1. Distributed N-Tier Client Server Architecture

3.1.1. N-Tier Architecture

The classic definition of a 3-tier architecture is 'the partitioning across the client (tier 1), the application server (tier 2) and the database (tier 3). In the 3-tier model the application logic lives in the middle tier and is separated from the user interface and the data.

The introduction of high speed networks, the Internet, mobile clients, and application servers have blurred the line between strict client and server. In many cases clients, web servers and application servers maintain their own local data store and switch roles between client and server depending upon the situation. In this environment client-server is more of a conceptual pattern that applies on a transaction by transaction basis at runtime, thus N-tier (Figure 4).

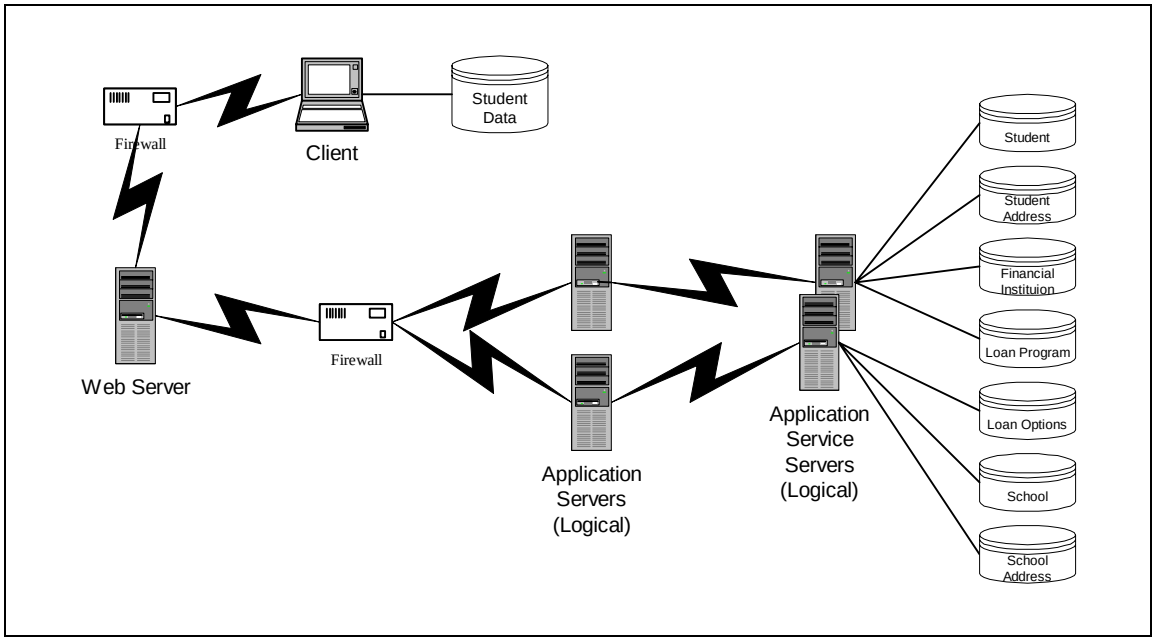

Figure 4 - N-Tier Client Server Architecture

3.1.2. Client Server Architecture

Client Server Architecture defines a set of building block (components) that can be used to develop a distributed system. These blocks are comprised of the Client, the Server and the middleware – the glue responsible for holding the client and server together. These components have the following characteristics:

- Client – The client uses the distributed services of the middleware to implement an interface to the server in order to obtain desired information stored elsewhere. The client is responsible for implementing presentation services that are appropriate to the end resource. This is most often a graphical user interface (GUI) such as a web browser.
- Middleware – The middleware runs in both the client and server environments. It provides essential common services used to implement network communication, transactions and security to name a few.
- Server – The server provides a given set of services. It depends on an interface between the native operating system (OS) and the middleware to process requests for a service. The server provides the services for accessing data and business logic.

3.1.3. Distributed System

A distributed system provides applications with the ability to execute segments of their logic on different hardware platforms using different operating systems in possibly different physical locations. With the advent of powerful multiprocessor based mini-computers and the establishment of high-bandwidth networks, distributed computing has become a viable implementation strategy. Some benefits of a distributed computing environment are:

- Increased Reliability – redundant workload managed servers provide protection against system crashes.
- Increased Scalability – workload is effectively scaled by adding additional servers.
- Increased Performance – by viewing the 'network as a computer', distributed applications are capable of leveraging resources across the network.

3.2. 'Model View Controller' Architectural Pattern

The Model-View-Controller (MVC) architectural pattern divides an interactive application into three basic components. The model provides the business logic and interface to data stores. Views provide a client specific implementation of the data obtained from the model. MVC architectural pattern maps exceptionally well to a distributed client-server system. Moreover, each layer of the MVC maps well to a component based implementation. This clear separation of the architectural components allows for independently designed and developed modules that are bound at runtime. Figure 5 shows the architecture of an Internet based thin client implementation of the MVC pattern. Note the MVC pattern is not limited to a thin client implementation nor is its use restricted to end user clients. Furthermore, the physical location of the MVC layers is not relevant.

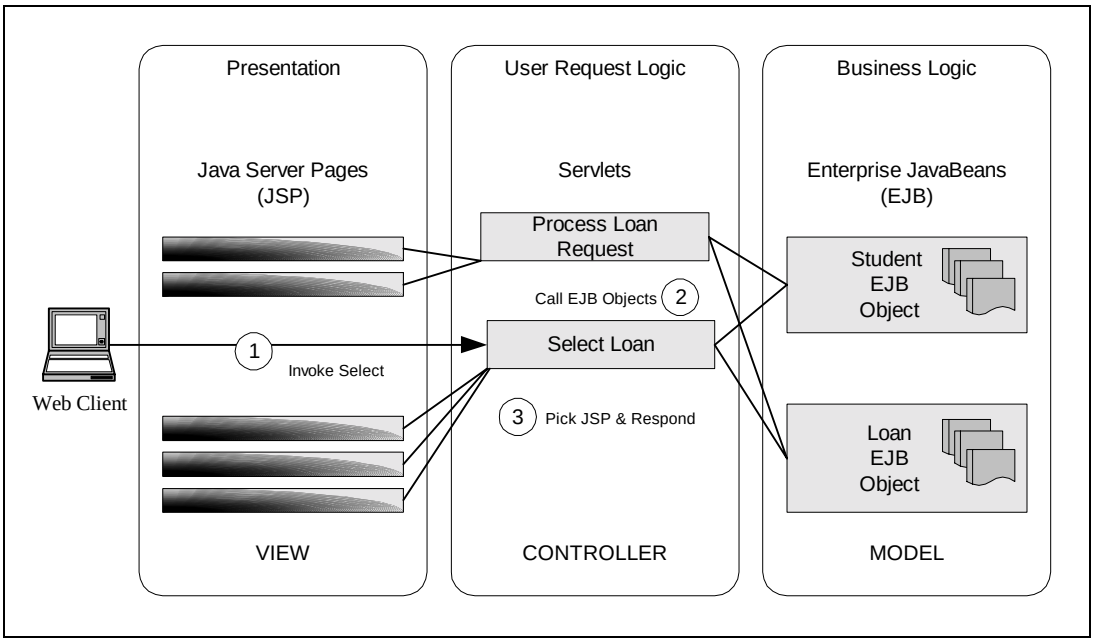

Figure 5 - Model View Controller Pattern

3.2.1. Advantages of the Model-View-Controller Pattern

The advantages of using the MVC Pattern in the SFA architecture provides the following benefits:

- Multiple views of the same business model – Since MVC provides for a strict separation of the business model from its representation, multiple views across heterogeneous delivery devices, can therefore be implemented and used with a single model.
  - Synchronized view – The change propagation mechanism of the MVC pattern ensures that changes to the model are synchronized with all attached observers at the correct time.
  - Pluggable views and controllers – The logical separation of the model, view and controller will allow Department of Education to exchange the view and controller of a model as the business requires.
  - Interchangeable views – The MVC will allow for the implementation of many views of the same model with respect to such constraints as user access levels.
- Internet fit – An MVC pattern is an excellent fit in the Internet environment where there is a distinct separation between the end user view and the business model. Figure 5 depicts the separation of the view from the model by using a web browser as the client and EJBs to model the business components.
- Separation of responsibilities – Separate development teams can be assigned to implement the different functional areas of the MVC.

3.3. Fat Client vs. Thin Client – The Advantage of The Web Browser Client

The term Fat Client / Thin Server defines a strategy for placing the bulk of the processing and business logic within the client of a client server implementation. Likewise, Thin Client / Fat Server defines a client server architecture in which the bulk of the processing and business logic is provided by the server. Either implementation requires that the client be responsible for the majority of the end user presentation interface.

Over the past several years the increasing functionality of web browsers has made thin client development a technological reality. Providing content via the web has reduced the need to develop and distribute expensive client applications. Even so, until recently developing applications with complex formatting and processing via the web required the development of an applet. Now complex formatting and business logic can be implemented by using a mix of servlets, JavaServer Pages (JSP) and Java Beans. This alleviates the need to consider the latency time and bandwidth required to download an applet for use in a browser.

3.3.1. Web Browser Client Advantages

The following is a list of advantages that a thin client holds over the use of a fat client.

- Thin clients provide a greater level of scalability because the fat server can be workload balanced. The client platform is usually the limiting factor with respect to processing power. Adding additional severs is easier than replacing client machines.
- Thin clients reduce the costs of development and deployment – there is little to deploy to client machines other than the browser.
- Web based thin clients provide platform independent client access to applications.
- Thin clients do not require specific versions of browser based implementations with the exception of the most recent Hypertext Markup Language (HTML) support.
- A true web based thin client does not require the same long latency periods for downloading as do applet based fat clients.

4 The ITA Conceptual Architecture

4.1. The SFA Architecture Domains

The ITA is based on three core architecture domains within the SFA technical environment, which are targeted at reducing stovepipe systems, islands of technology and the need for customized point-to-point system interfaces. The three new domains include the Internet Domain, the Enterprise Application Integration Domain, and the Enterprise Data Domain. These three domains combined with the current Legacy Domain, make up the SFA enterprise and the ITA. The business rules, integrity checks and sequence of steps associated with a business function are implemented in a logical black box referred to as a 'service.' Services provide a set of published interfaces that allow participating applications to extend their business processes.

Together the four domains of the ITA provide the necessary infrastructure to implement a service-oriented architecture. However, in order to provide this infrastructure each domain must provide support for seamless integration between domain touch-points. Each domain must provide a set of interfaces or connectors that allow integration with the services of intersecting domains. For example, the Internet Domain must provide interfaces to the Data Domain in order to provide persistence for stateful business objects. The Data Domain must provide adapters to disparate data sources in order to feed the extract / transform / load process. The Enterprise Architecture Integration (EAI) Domain must provide connectivity to different legacy architectures and data sources in order to provide integration with existing SFA systems.

Figure 6 shows how the four domains overlap to provide seamless integration between domain services. The EAI layer is the glue for providing the integration between the domains where interfaces do not exist. The EAI layer provides a set of application adapters and communication services that can span domains thereby providing additional integration points between domains.

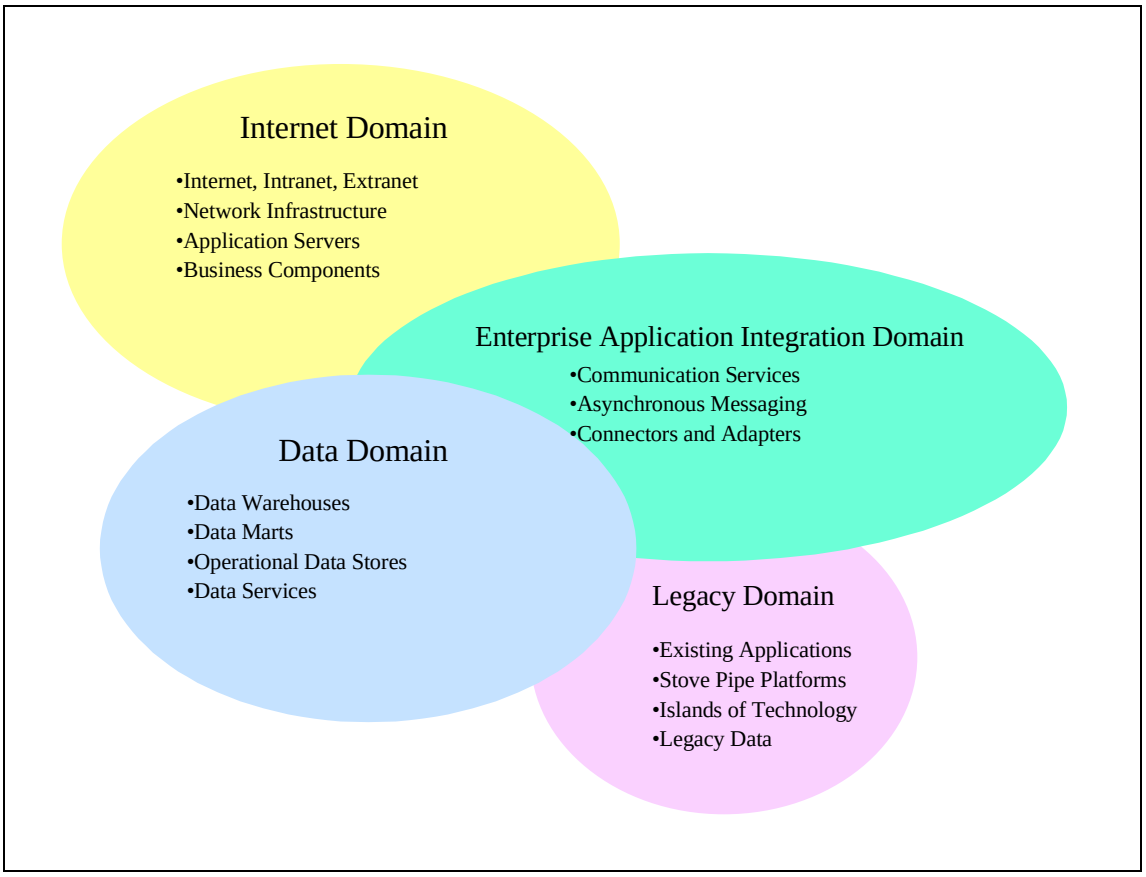

Figure 6 – The Four Domains of the SFA Architecture

The successful implementation of the ITA domains will provide the infrastructure for integrating COTS packages with other SFA systems. Using the integration interfaces supplied by the domains will make it possible to avoid the creation of system silos that are not part of the overall business solution. The ITA provides support for the integration of new COTS packages into either the Internet, EAI or Enterprise Data Domain. The ITA supports the integration of COTS packages that support the following domain interfaces or frameworks:

- Enterprise Java Beans (J2EE)
- CORBA Business Objects
- Work Flow Management
- Asynchronous Messaging (AMI, CMI, JMS, etc)
- Data Warehouse Analysis (ROLAP, MOPLAP, etc)

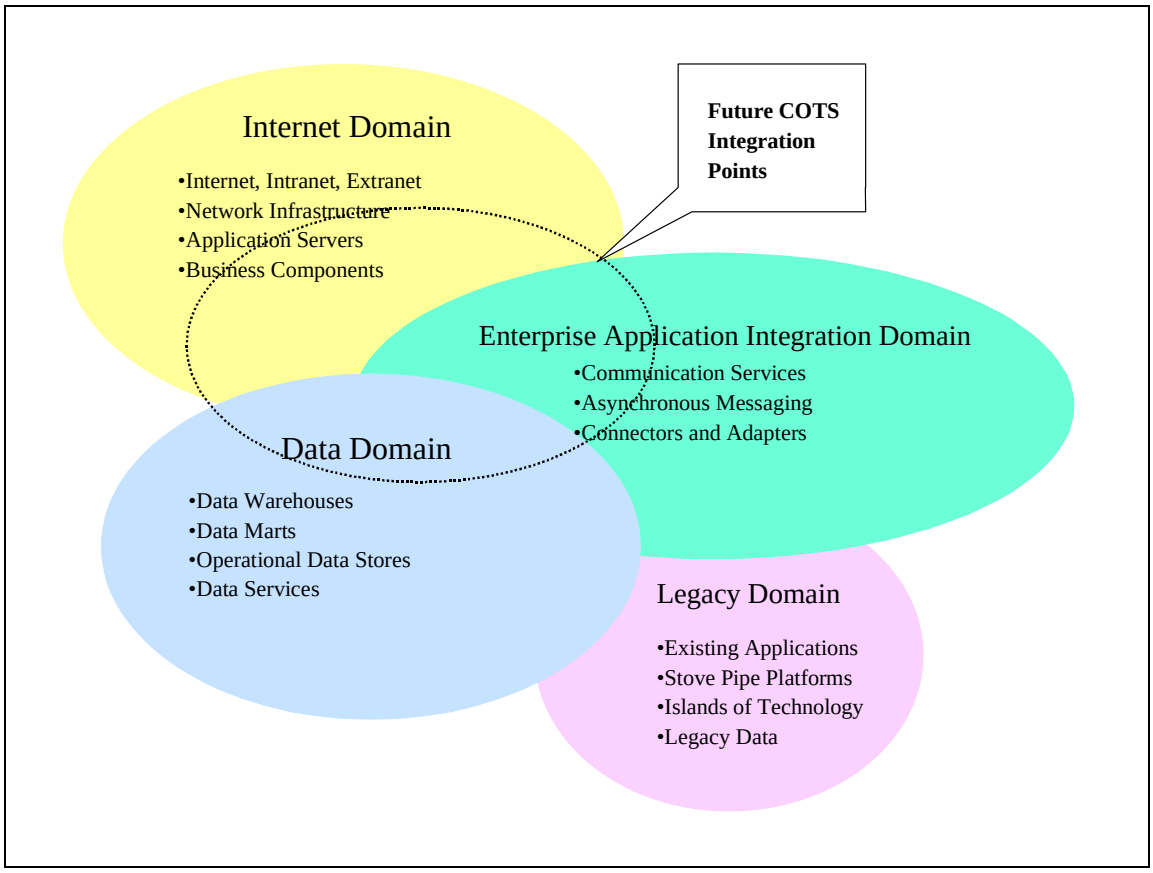

Figure 7 – Future SFA COTS Integration Points

4.2. Product to Domain Mapping

The Department of Education's SFA Modernization Task Order 4 (TO4) defined the requirements for implementing a Service-Oriented Architecture. TO4 also selected the vendors and products that provide the functionality required for implementing the necessary functionality with each domain of the ITA. Table 2 provides a map for each selected product to the target domain according to function.

Table 1 – Integrated Technical Architecture Product to Domain Map

Product Type	Product Name	Function	Domain	Domain Interfaces
Web Server	IBM HTTP Server	- Thin Client Presentation - Render HTML	Internet	Enterprise Data
Web Application Server	IBM WebSphere Advanced Edition	- Servlets - JSPs - Web Security	Internet	Enterprise Data, EAI

Product Type	Product Name	Function	Domain	Domain Interfaces
Business Object Server	IBM WebSphere Enterprise Edition Component Broker	- EJBs - CORBA - Adapters	Internet / EAI	EAI, Legacy, Enterprise Data
Portal Server	Viador Portal Server	- Portal	Internet	
Search Engine	Autonomy Search Engine	- Web Spidering - Content Searching	Internet	Enterprise Data
Content Management	Interwoven TeamSite	- Content Management - Configuration Management	Internet	
Internet Load Balancing	IBM WebSphere Performance Pack	- HTTP Spraying - IP Redirection - Load Balancing - Caching	Internet	
Directory Server	Netscape LDAP Server	- Directory Services - Privilege Security	Internet / EAI	EAI
Message Oriented Middleware	MQSeries, MQSeries WorkFlow	- Asynchronous assured message delivery - Application Adapters - Business Process Management	EAI	Internet, Enterprise Data, Legacy
Message Broker	MQSeries Integrator	- Message Transformation and Routing	EAI	Enterprise Data, Legacy
Data Warehouse Management	Informatica	- Data Extraction - Data Transformation - Data Loading	Enterprise Data	Legacy
Data Warehouse Analysis	Microstrategy	- OLAP - ROLAP - MOLAP	Enterprise Data	Internet

4.3. SFA Domain Architecture Topologies

This section describes the topology of each ITA Domain. The execution topologies required to support the ITA Domains will be shown in graphical form in the following sections. Each topology diagram will consist of several nodes that represent the function of that node. The execution topology of each SFA Domain Architecture will be described and mapped to a set of vendor products.

4.3.1. The End-to-End Architecture Topology

The following diagram provides a high-level view of the SFA Service-Oriented Architecture. The diagram depicts the run-time topology necessary to support all of the architectural domains. Some minor elements of each domain may be excluded to allow for diagrammatic elegance.

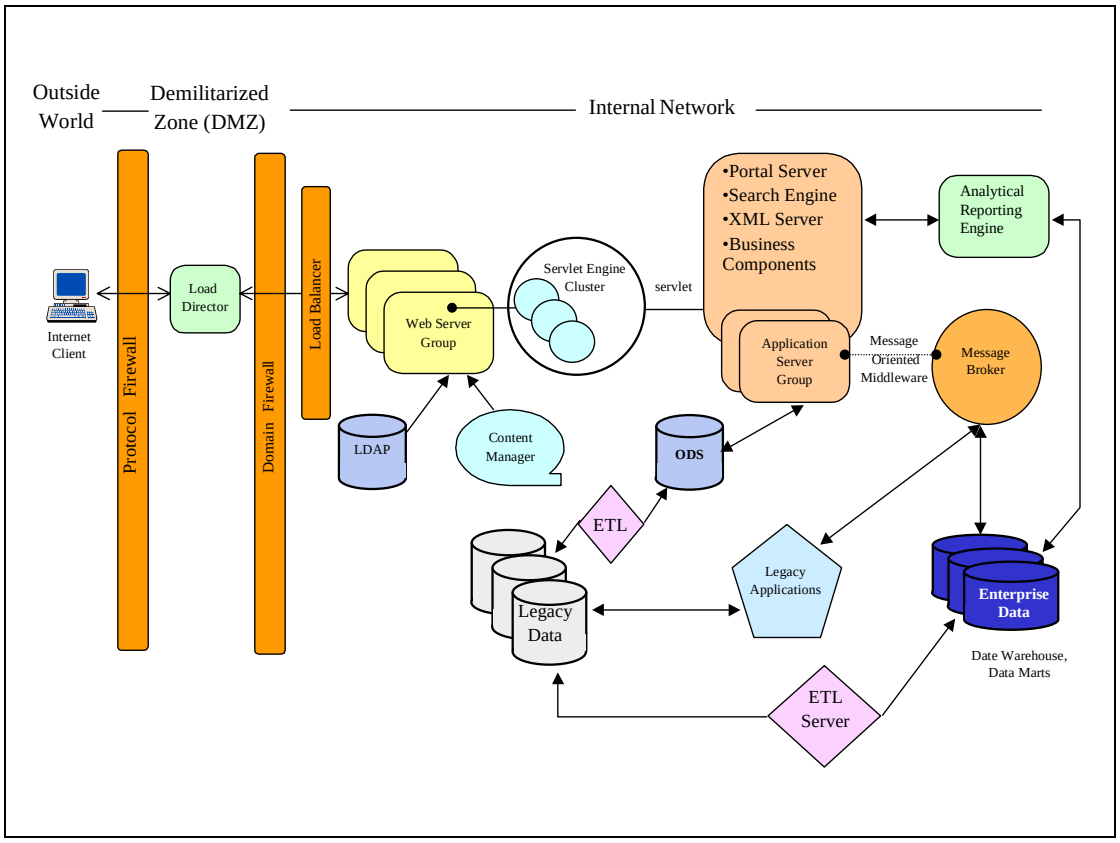

Figure 8 – End-to-End ITA Topology

4.3.2. Internet Architecture Topology

The Internet Architecture (Figure 9) supports the delivery of web-based applications to SFA's users on the World Wide Web (WWW). Specifically, the execution topology in Figure 9 provides the infrastructure necessary to support the delivery of web based thin-client

applications. The topology defined in this section defines a highly scalable and available feature rich enterprise internet architecture. The implementation of this architecture within SFA will be deployed in stages according to application or system requirements. For example, the implementation of LDAP and security servers will be deployed in the later stages of the implementation of the technical architecture.

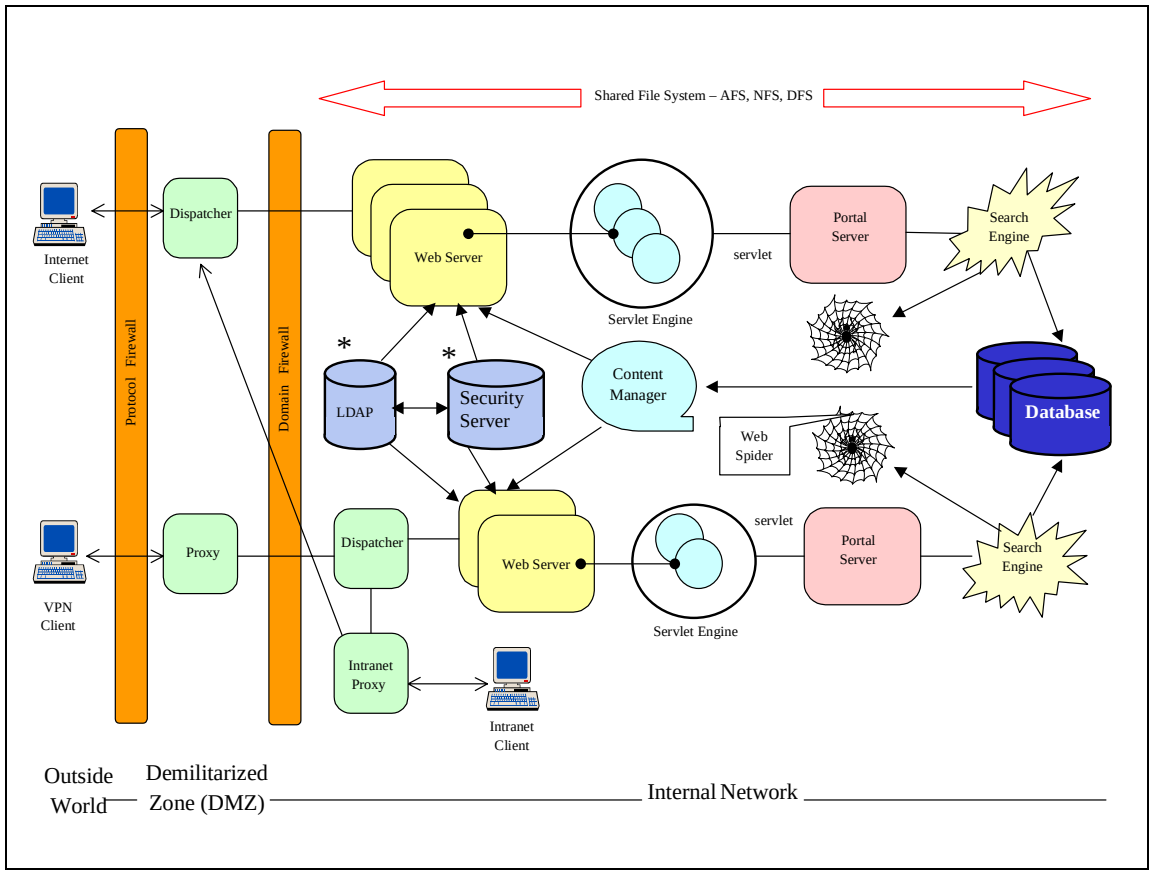

Figure 9 – Internet Architecture Topology

Note: Elements that are prefaced by a * will be implemented in the later stages of SFA technical architecture.

The Internet Architecture includes the following components or nodes:

- **Protocol and domain firewall nodes** – Firewalls provide services that can be used to control access from a less trusted network to a more trusted network. A protocol firewall provides screening routers according to protocol type. Domain firewalls provide application gateways to applications located within the enterprise.
- **Load balancing and caching node** – The load balancer provides horizontal scalability to the web servers by dispatching Hypertext Transfer Protocol (HTTP) requests among several identically configured web servers. This node may also provide caching for frequently accessed web pages.

- **Shared file system node** – The synchronization of content and file system access authority is achieved by using a shared file system thereby capitalizing on the replication capability of this technology.
- **Web Server Node** – The web server node is an application server that includes an HTTP server and is typically designed for access by HTTP clients and to host presentation logic. The web server also provides services for serving other protocols like FTP, sound and streaming video.
- **Web Application Server Node** – This node provides the infrastructure for component based application logic. The web application server(s) provide the underlying services for running servlets, JSPs, EJBs and CORBA objects.
- **Search Engine Node** – Provides services for searching data and returning uniform resource locator (URL) links to data that matches the provided search criteria.
- **Content Management Node** – This node manages the static content that is accessible through HTTP links. The content must be managed according to conventional configuration management techniques.
- ***Directory and Security Services Node** – These nodes supply information on the location, capabilities, access privileges and various attributes related to resources and users known to the enterprise. This node supplies security services for authentication and authorization to enterprise resources according to users privileges.
- **Portal Server Node** – The portal server node provides the services to dynamically define a standardized enterprise look and feel to web applications. The portal server also provides the user with the ability to personalize web applications according to their preferences.

Further decomposition and explanation of the Internet architecture may be found within the Scenarios sections as well as in the section that defines the Internet Architecture Detailed Design.

The following topology diagram depicts the Internet Domain Architecture product mapping designed to support the content searching through the SFA portal.

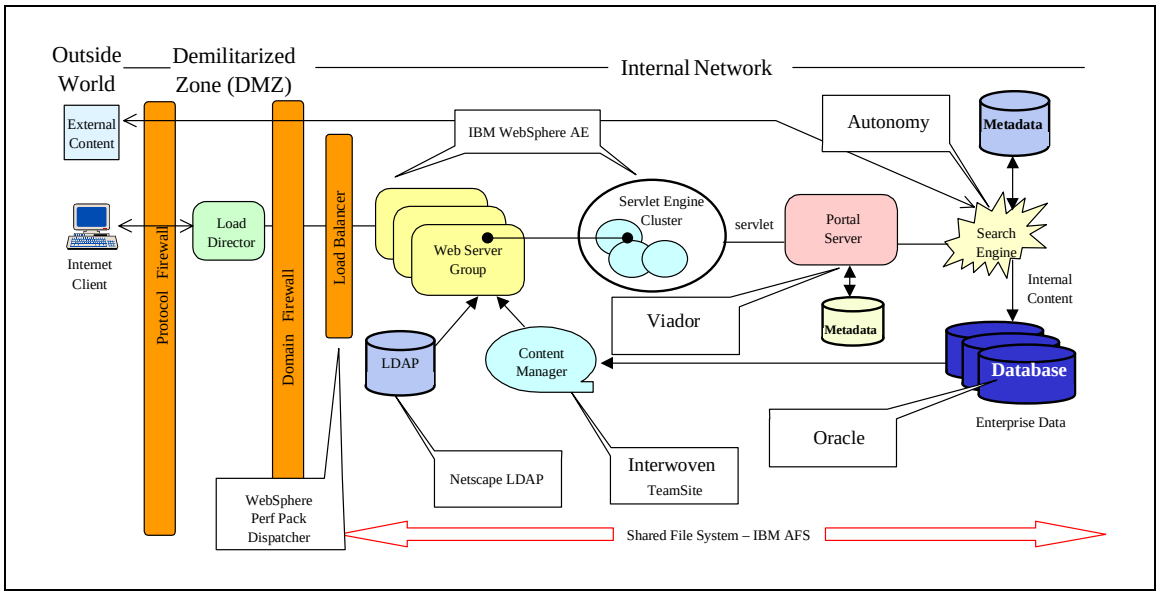

Figure 10 – ITA Internet Domain Architecture Product Mapping

4.3.3. The EAI Architecture Domain Topology

The Enterprise Application Integration (EAI) Domain Topology provides the services necessary to support application integration across the SFA Enterprise. The topology supports the implementation of an asynchronous messaging back-plane that forms the basis for the deployment of a hub and spoke network architecture. The topology also supports application connectivity and transaction processing through the integration of pre-built application adapters, gateways and connector frameworks.

The Enterprise Application Integration Domain Topology includes the following components or nodes:

- Queue Managers – Queue Managers are run-time processes that are responsible for managing queues of messages for application programs using International Business Machine's (IBM) MQSeries. Queue Managers help maintain the flow of messages through a queue and manage communication links between other queues.
- Message Broker -The role of the Message Broker is to provide a layer of logic to the messaging layer. The Message Broker is the 'glue' between intercommunicating applications. The Message Broker ensures that messages are routed correctly between applications and in a format in which the applications can understand each other. SFA applications will use the Message Broker MQSeries Integrator (MQSI) to provide a mechanism for:
  - intercommunication between applications that are served by the application server
  - asynchronously obtaining data from legacy applications
  - asynchronously updating legacy applications
  - routing data to and from Enterprise Data Warehouses and Data Marts.

- WorkFlow Server – The WorkFlow node uses the messaging infrastructure for communication. The role of a WorkFlow Server is to automate business processes involving people and applications to give the enterprise added control over business activities. In the case of SFA it allows the developer to establish rules for managing updates to applications across the enterprise.
- Application Adapters - Application Adapters provide direct connectivity to applications and disparate data sources. Application Adapters allow application to transactionally coordinate updates across different data sources. Application Adapters are used by Application Servers such as Component Broker (CB) to provide support for On-Line Transaction Processing (OLTP).

The following topology diagram depicts a possible execution topology and product mapping for the SFA EAI Architecture Domain.

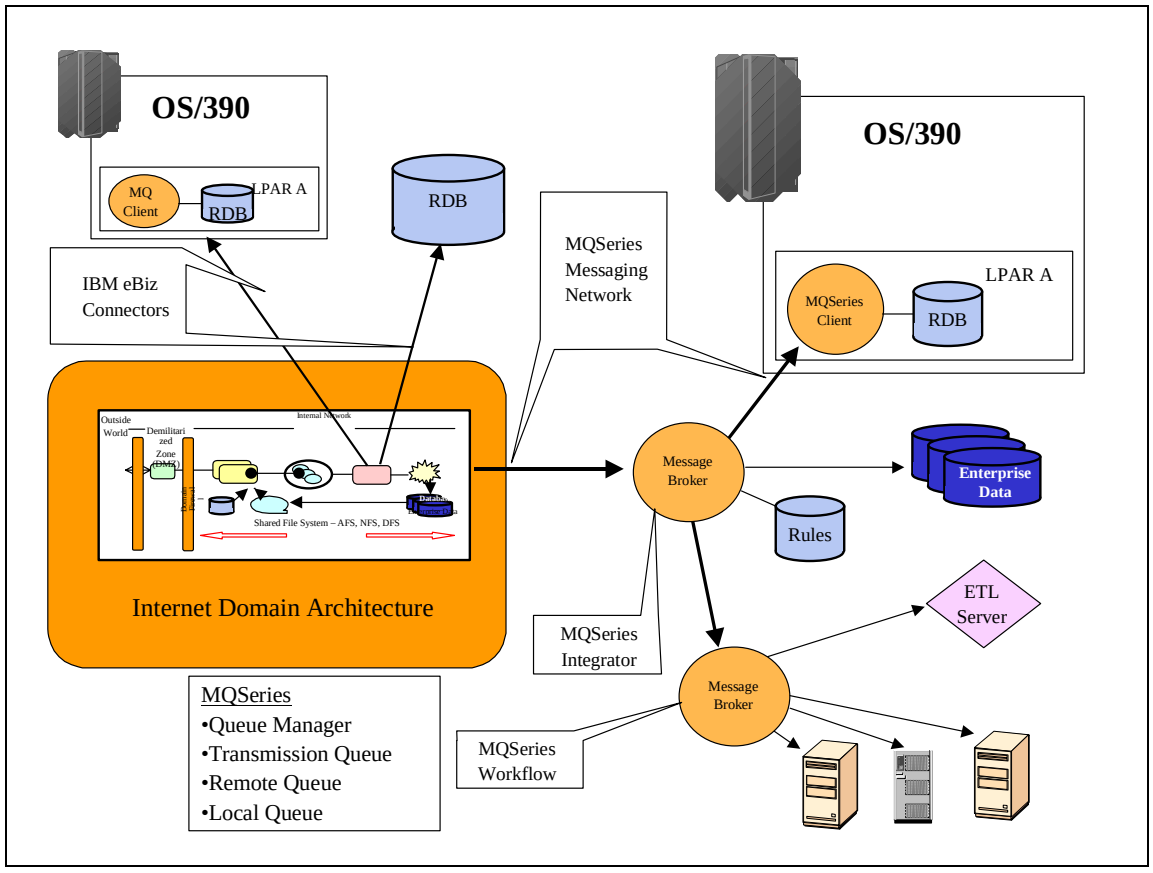

Figure 11 - ITA EAI Domain Architecture Product Mapping

4.3.4. The Enterprise Data Domain Topology

The data layer consists of two segments in the Department of Education SFA technical architecture, the legacy data domain and the Enterprise Data Domain.

Legacy Data Domain

The Legacy Data Domain (LDD) is based on existing data stores used by the current legacy infrastructure. Legacy applications and the LDD comprise the Legacy Domain. The majority of information stored in the legacy data domain is backed by DB2 on the OS/390. Direct access and acquisition of data stored in DB2 is not possible because the database schemas are not normalized and highly coupled to their respective applications. Data stored in the legacy data domain must be considered a secondary data repository once its controlling application is deprecated or duplicated.

Enterprise Data Domain

The Enterprise Data Domain (EDD) is the data repository that will be supported by future enterprise applications. The Enterprise Data Domain is comprised of data warehouses, data marts and operational data stores. The EDD is accessed and updated by applications that live in either the Internet or EAI Domain. The data stored in the EDD will be considered to be the 'official' copy of Department of Education's corporate data. The EDD provides Department of Education with a more normalized data schema that lends itself to the use of business components and object relational mapping. The EDD will be updated in real-time while the LDD will have some period of update latency. This will be true only when a new infrastructure application is responsible for controlling updates to the same logical schema in the EDD.

The SFA Enterprise Data Domain Topology includes the following components or nodes:

- ETL Node – The Extract Transform Load (ETL) node provides the services necessary to perform the Extract, Transform, Load and Distribute processes used to create the Data Warehouse, Data Marts and Operational Data Stores.
- OLAP Node – The On-Line Analytical Processing (OLAP) nodes role is to provide a real time reporting engine capable of analyzing the large data sets usually associated with a data warehouse.
- Database Node – The database node provides support for implementing a centralized database that collects, organizes and stores data from SFA's operational systems to provide a single source of integrated and historical data for the purposes of the end-user reporting, analysis and use in decision support and Customer Relationship Management (CRM) systems like Seibel.

The following diagram depicts a possible execution topology and product mapping for the SFA Enterprise Data Domain Topology.

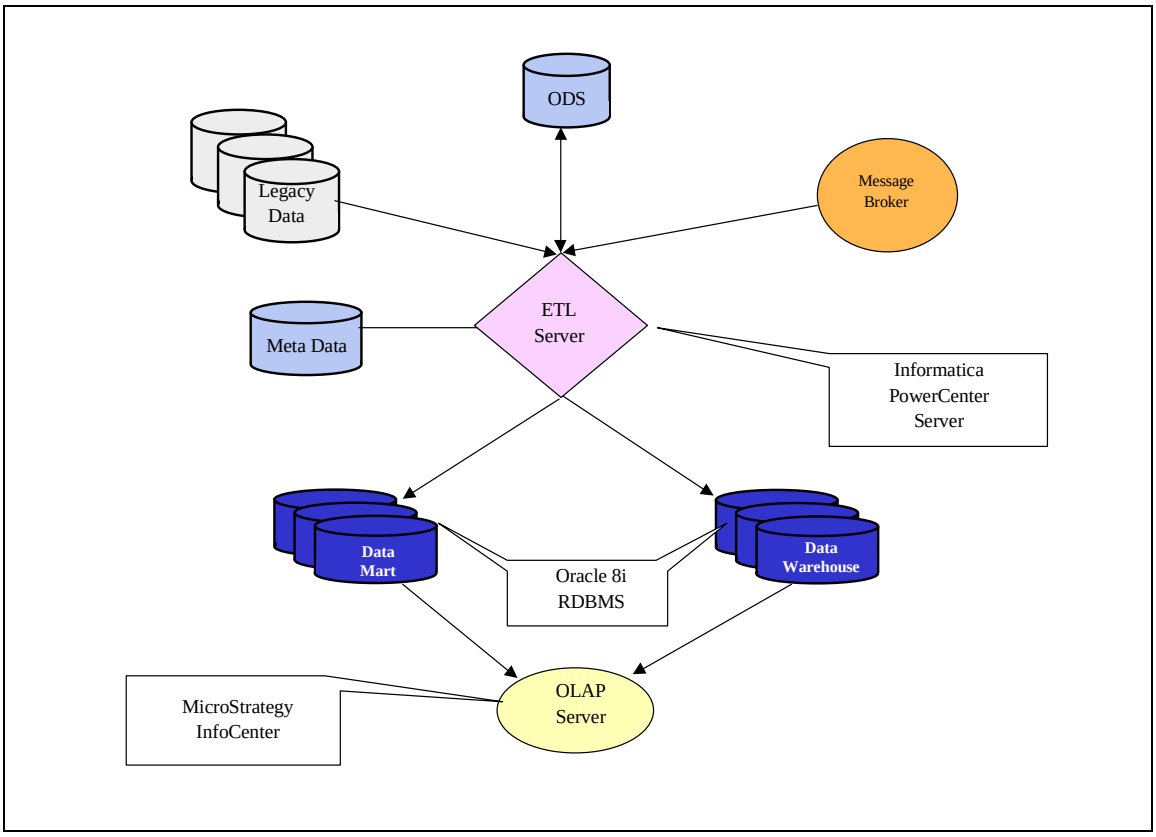

Figure 12 - ITA Enterprise Data Domain Architecture Product Mapping

5 Technical Architecture Scenarios

This section defines the scenarios that will occur with in Department of Education's technical architecture. Each scenario is a section of the proposed technical architecture that as been partitioned into interrelated slices. These sections lend themselves to further technical decomposition and are focused on a particular architectural theme. The defined scenarios may overlap or work in conjunction with another scenario.

5.1. Overview of Technical Architecture Scenarios

The following table describes each of the architectural scenarios, the layers that are involved and assigns a number to the scenario for reference purposes.

Table 2 - Technical Architecture Scenarios

Scenario #	Description	Presentation Layer	Internet Domain	EAI Domain	Enterprise Data Domain	Legacy Domain	Client Type
1	Thin Client Accessing Business Application Components (No Legacy)		X	X	X		Thin Browser
2	Batch Process Accessing Business Application Components	X	X	X	X		Batch
3	Business Application Components Coordinating Legacy Updates		X	X	X	X	
4	Legacy Applications Coordinating Business Application Components		X	X		X	
5	Searching Content Using the Internet Portal	X	X	X	X		Thin
6	Data Population Using an ETL Process				X	X	Batch
7	Coordinating Transaction Processing	X	X	X	X	X	Any
8	Accessing Applications Through The DMZ	X					Thin

5.1.1. Scenario #1: Browser Based Thin Client Accessing Business Application Components

Purpose

This scenario defines an Internet based thin client architecture. The goal of this architecture is to provide the end user with an interactive GUI that accesses business application

components. The GUI is implemented using a Java enabled web browser like Netscape Communicator.

Architectural Pattern(s)

Scenario #1 is based the following architectural patterns:

- Model View Controller
- CORBA Services
- Enterprise Java Beans
- Web Servlets
- JavaServer Pages (JSP)

Development Languages

The following table defines the languages and the purpose for using them with in this scenario.

Table 3 - Scenario #1 Programming Languages and APIs

Programming Languages and APIs	Purpose(s)
Hypertext Markup Language – HTML	Develop the presentation of the web page. Define the location of the web server. Provide context for executing servlets on the web server.
Java	Develop business components Develop servlets Develop JSPs
C++	Develop CORBA business components invoked by other business objects such as CORBA JavaBOs and EJBs.

Scenario Protocols

The following table lists the different protocols used to implement scenario #1.

Table 4 - Scenario #1 Protocols

Protocols	Purpose(s)
HTTPS – Secure Hyper-Text Transport Protocol	Used to send secure information to and from the web client and web server. Provide encryption of HTML to and from browser and web server.
HTTP – Hyper-Text Transport Protocol	Used to send information to and from the web client and web server.

Protocols	Purpose(s)
IIOP – Internet Inter-Orb Protocol	Provide network communication for CORBA and EJB objects.
TCP / IP – Transmission Control Protocol / Internet Protocol	Network transport protocol used from client to web server

Related Architectural Scenarios

This scenario is closely related to the Demilitarized Zone (DMZ) architectural scenario. The DMZ scenario outlines an architectural pattern used to define secure Internet access to enterprise resources. This scenario assumes direct access from the Internet client to the web page. The flow of this scenario is applicable to that of an Intranet application.

Scenario Assumptions

The following is a list of assumptions, which help to define the state of the system before the scenario begins:

- An application that uses HTML is served by a web server. This server is capable of executing servlets and Java Server Pages. Support for this requirement is provided by the WebSphere Advanced Edition (AE) servlet engine.
- Access to the web server via the firewall is not considered part of the scope of this scenario. See the discussion on the DMZ pattern.

Operational Flow of Scenario #1

This section defines the basic processing flow required to provide thin client access to business application components (Figure 13).

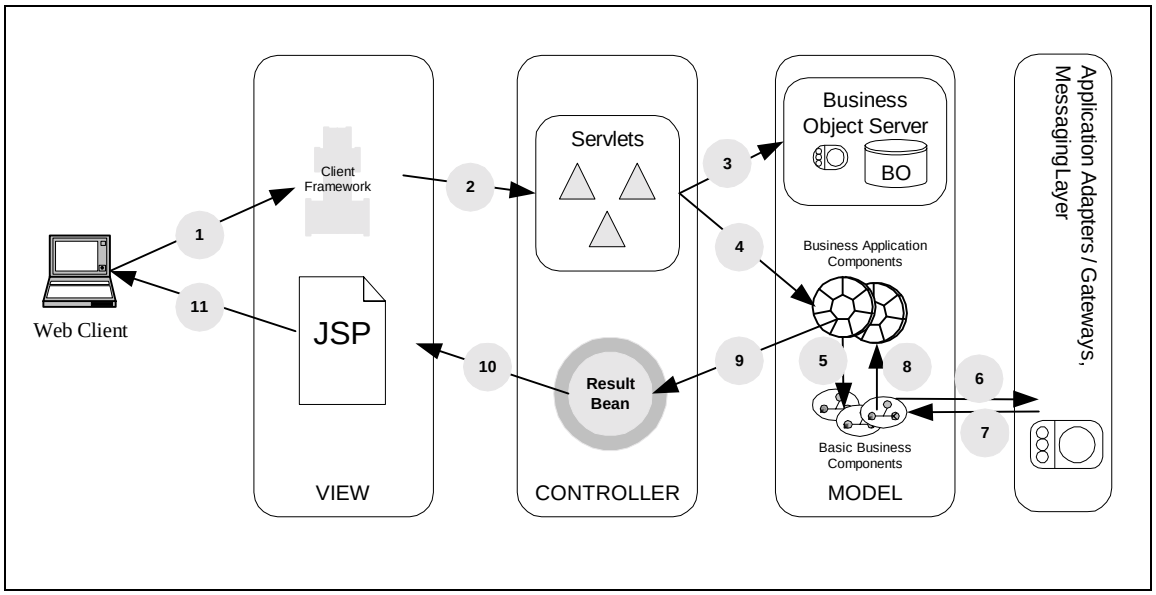

Figure 13 - Scenario #1 (Thin Client Accessing Business Application Components)

1. The user selects a link on the web page that instructs the web application server to execute a servlet. The servlet defined in web page has been registered with the web server. This servlet is part of the client framework and knows how to present information defined by the user to a controller servlet. For example, the servlet may be invoked as a result of a <form method=get action="servlet name"> HTML tag. However, there are numerous ways to invoke servlets. The initiating servlet may pass control to another servlet registered in the application server. This is called servlet chaining and is useful when the output of one servlet is used as input for another. One use of this technique may be to limit the number of queried items returned to the browser. The client web browser communicates to the web server using HTTP or Secure Hyper-Text Transport Protocol (HTTPS) depending on the security requirements. Secure HTTPS should be used when users exist outside the trusted network and are accessing sensitive data.
2. The servlet that coordinates the processing of business components is called the *controller*. This servlet lives in the application server and is responsible for presenting the request to business components and processing the results.
3. The controller servlet finds the business component server and sets up communication between the servlet and the server in order to begin processing the request. In the case of EJBs the controller uses the Java Naming and Directory Interface (JNDI) API to access the location of the server that provides the necessary services. This is done by the servlet which provides the namespace location and context factory class that are placed into the bean properties. A network connection to the root of the namespace is then established by invoking the InitialContext(props) method. The namespace provides the location of the Home interface that will be used to find instances of business components. Several options exist for determining the location of the naming server or location of a particular object within the naming and directory service. Options include, storing the information in LDAP, properties files or metadata tables in a database.
4. After obtaining a reference to the necessary business application component the controller invokes a message to the component that will process the desired request. Messages to EJB application components are invoked using the Remote Method Invocation (RMI) API. However, the underlying network implementation may be different for each EJB vendor. For example, IBM's Component Broker uses RMI / Internet Inter-ORB Protocol (IIOP) as the underlying transport protocol.
5. The business application component obtains references to a set of basic business components necessary to process the request from the application component. This entails querying the application server using Object-Oriented (OO) Structured Query Language (SQL) to obtain either a collection of object references or a set of database tuples. Object references are useful when additional processing by the business object is necessary to complete the request. Returning a set of database tuples is useful when presenting query sets to a requester. The business application component is responsible for defining and coordinating the transactional context. Business application components can be either CORBA or EJB based – it does not matter.

6. Basic business components utilize an application adapter to Oracle (distributed) or DB2 (mainframe) to issue request to the database to obtain specific tuples in the database, which correspond to the object relational mapping of the basic business component.
7. Data that matches the queries issued by the basic business components is returned to the application adapter and is used to populate instances of the corresponding business component.
8. The application business component processes the basic business components according to their business logic.
9. The application business component returns the results of the request back to the controlling servlet via a result bean. A result bean defines the structure of the data to be passed back to the servlet according to the request.
10. The controlling servlet uses a predefined JSP that lives in the application server to re-present pre-formatted results to the web browser. The JSP defines a template for how the HTML page will be displayed.
11. The JSP creates a dynamic web page from the results and sends the response back to the client via the web server. The dynamic web page is created according to the presentation template defined as part of the JSP.
12. The web server processes the doPost() issued to the client and sends the web page created by the JSP back to the web browser.

Benefits of Scenario #1

The greatest benefit of scenario #1 is that it provides a very thin client with access to enterprise resources. This reduces the need to consider client configuration during application development or deployment. Additional benefits are:

- Greatly improved portability. Because servlets are written in Java, they are portable across platforms, so that they do not have to be recompiled for different operating systems. The servlet interface, being a standard, allows servlets to be moved from one servlet engine to another, as long as the servlets do not use vendor extensions.
- Separation of business logic from presentation logic. This allows for greater reuse of business logic across different clients.
- Makes implementation of the DMZ architectural pattern easier. Separation of the web and application servers restricts the flow of IIOP and other enterprise specific network protocols (such as SNA) to with in the trusted network.
- No direct access to business components thereby adding a degree of security.
- Promotes load balancing of the physical layers – web server, application server and the database server. Each physical layer can be work load balanced independently providing for a highly scaleable and robust computing environment.

Liabilities of Scenario #1

The liabilities of implementing applications that utilize scenario #1 are as follows:

- The development of a web based MVC may be more complex than a traditional centralized 2-tier client server application. Determining the right granularity between the controller and the model reduces the implementation complexity.
- The implementation of the client, the servlet, and the business components are likely to be developed by different staff members or teams. A thoroughly documented analysis and design is required before development begins to avoid unnecessary defects.
- Changes to the business model components may require changes to the controller servlet as well as the JSP which re-presents data.
- Servlets are web centric controllers and as such can not be used in other MVC implementations. The duplication of functionality within the system that mirrors a web based MVC implementation using another paradigm (fat client) will require the re-development of the controller functionality in the new paradigm.

5.1.2. Scenario #2: Batch Applications Accessing Business Application Components

Purpose

This scenario describes the basic flow of an application that uses a fat client to manipulate business components to implement a business process. Fat clients may include applications with or with out end user interfaces. Batch applications are fat clients with very thin user interfaces. Applications that use applets or other client frameworks to provide an end-user with a GUI interface are fat clients with a thick user interface.

Architectural Pattern(s)

Scenario #2 is based the following architectural patterns:

- Model View Controller
- CORBA Services
- Enterprise Java Beans
- Java Applets / GUI Frameworks

Development Languages and Application Programming Interfaces

The following table defines the languages and the purpose for using them within this scenario.

Table 5 - Scenario #2 Programming Languages and APIs

Programming Languages and APIs	Purpose(s)
Java	Develop business components Develop Controller access to business components. Develop GUI interface using static or web based applets.
C++	Develop CORBA business components invoked by other business objects such as CORBA JavaBOs and EJBs. Provide fat GUI clients.
Applet API	Used to develop fat client GUI interfaces with Java.
Visual Basic	Quick development of MS-Windows / NT based fat client applications.

Scenario Protocols

The following table lists the different protocols used to implement scenario #2.

Table 6 - Scenario #2 Protocols

Protocols	Purpose(s)
IIOP – Internet Inter-Orb Protocol	Provide network communication for CORBA and EJB objects.
TCP / IP – Transmission Control Protocol / Internet Protocol	Network transport protocol used from client to web server.

Related Architectural Scenarios

The business component infrastructure and client proxy framework mentioned in scenario #1 can be reused in scenario #2. The primary difference between the two scenarios is the implementation of the GUI and controller frameworks.

Scenario Assumptions

The following assumptions help frame the initial state of the application before the scenario begins:

- By definition the *view* and *controller* objects must be residents of the same application when implemented as part of a batch client.
- A batch client may reuse any or all of the business components used by a thin client.
- The scenario is based on an application with a Java applet graphical user interface. However, the same sequence of events would apply to other fat client implementations.
- The business components used in this scenario are implemented using CORBA objects. However, the overall flow of the scenario would be similar using Enterprise Java Beans.

Operational Flow of Scenario #2

This section defines the basic processing flow required to provide fat client access to business application component (Figure 14).

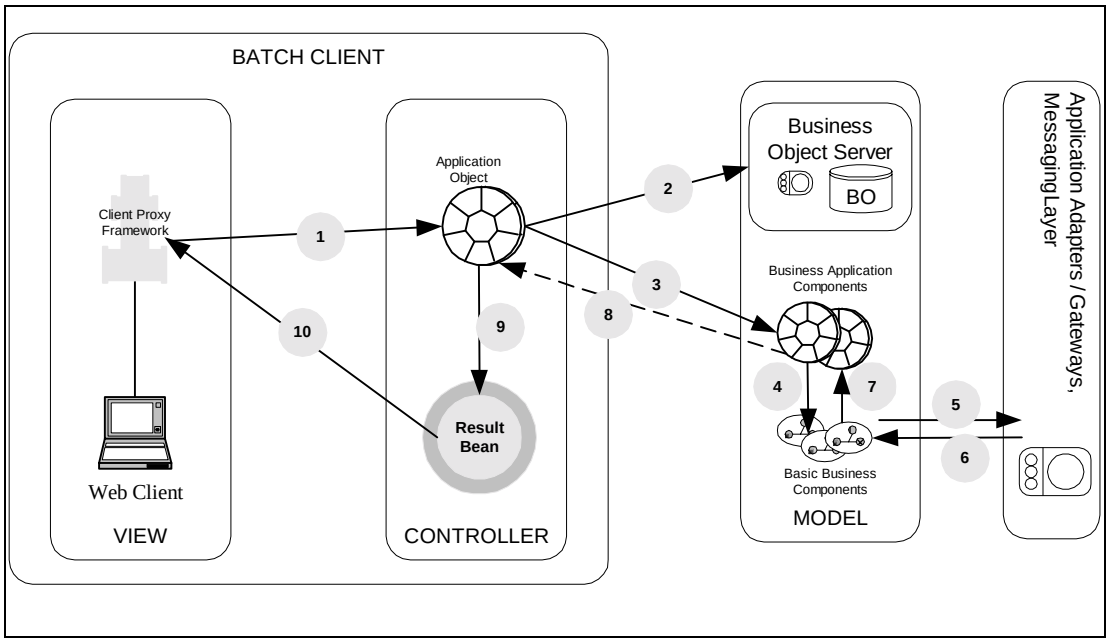

Figure 14 - Scenario #2 Batch Client Accessing Business Application Components

1. The user selects an option in the GUI interface that requires the services provided by a business application component in order to process the request. Once the GUI processing has been completed the *view* object passes the request to a *controller* for processing. The *controller* object resides in the same application as the *view* object. Both of these objects use the Java Bean component model to separate functional responsibilities. A *controller* that manipulates business components is called an *application object*.

2. The controller establishes communication with the business object server by initializing a connection to the Object Request Broker (ORB) and narrowing to the business object server that contains the necessary business component. This process is specific to the vendor implementation.

3. The *controller* obtains a reference to the correct business application component by navigating the name space via the CosNaming service and narrowing the reference to an interface. The applet (user interface) communicates with business components using the IIOP. Department of Education specific client proxies will provide clients with the ability to establish communication with business object servers without the overhead of having to repeatedly develop the same infrastructure code for every client. These proxies will be responsible for establishing and maintaining communication between the client and business object servers.

4. The business application component obtains references to a set of basic business components necessary to process the request from the application component. This entails querying the application server, using OOSQL, to obtain either a collection of object references or a set of database tuples. Object references are useful when additional processing by the business object is necessary to complete the request. Returning a set of database tuples is useful when presenting query sets to a requester. The business application component is responsible for defining and coordinating the transactional

context. Business application components can be either CORBA or EJB based – it does not matter.

5. Basic business components utilize an application adapter to DB2 to issue a request to the database to obtain specific tuples in the database, which correspond to the object relational mapping of the basic business component.
6. Data that matches the queries issued by the basic business components is returned to the application adapter and is used to populate the corresponding business component.
7. Basic business components manipulate the data returned from the adapter according to their business logic. Once processing is complete the information is presented to the business application component.
8. The business application component returns the results of the request back to the controller object in the applet via a result bean.
9. A result bean defines the structure of the data to be passed back to the client framework according to the request. A result bean defines the structure for passing the requested data back to the client.
10. The result bean is passed back to the applet and its contents are displayed to the user according to the logic in the *view* object.

Benefits of Scenario #2

The primary benefit of using a Batch client (in an n-tier model) is a reduction in processing pressure on the application server. Additional benefits are:

- Possible increase in application response time as compared to thin client.
- Allows for use of platform specific implementation techniques. For example, use of MS-Visual Basic programming for Windows-NT.
- Promotes the reuse of those architectural elements defined by Scenario #1.

Liabilities of Scenario #2

The liabilities of implementing applications that utilize scenario #2 are as follows:

- Fat client specific business logic may not be portable to other client platforms. Fat client business logic reuse has been traditionally low.
- Fat client performance requirements may out strip the client platform processor capability.
- Data sets returned from the 2nd tier to the client could become large and cause network bottlenecks.

5.1.3. Scenario #3: Business Application Components Coordinating Legacy Updates

Purpose

This scenario describes the process of coordinating legacy applications from business application components – Enterprise Java Beans or CORBA Business Objects. This scenario comes into play when business components have been developed to support the implementation of the EDD. Once an application that depends upon an EDD schema has been implemented, and mirrors functionality provided by a legacy application, the data stored in the EDD must be considered the 'primary repository.' For example, if a new member service is implemented that uses the EDD, updates to the member data (EDD and LDD) will be coordinated from the Business Component Layer.

Architectural Pattern(s)

Scenario #3 is based the following architectural patterns:

- CORBA Services
- Enterprise Java Beans
- Java Applets
- Forward-Receiver (Provided by MQSeries)

Development Languages and Application Programming Interfaces

The following table defines the languages, APIs, and the purpose for using them within this scenario.

Table 7 - Scenario #3 Programming Languages and API's

Programming Language and APIs	Purpose(s)
Java	Develop business components. Develop interfaces to and from the business components to the MQ interface.
COBOL / CICS	Develop legacy application CICS interface to be used for implementing the external legacy gateway.
CORBA Event API	Leverage the CORBA event service to coordinate updates from business components to the MQ legacy interface.
MQI, JMS, AMI	MQ Application Programming Interface – used to implement the MQ server to route updates to the legacy domain. Java and OAG messaging standards.
XML	Provide a common message format for data exchange and flexible message communication.

Scenario Protocols

The following table lists the different protocols used to implement scenario #3.

Table 8 - Scenario #3 Protocols

Protocol	Purpose(s)
IIOP – Internet Inter-Orb Protocol	Provide network communication for CORBA and EJB objects.
TCP / IP – Transmission Control Protocol / Internet Protocol	Network transport protocol used from client to web server

Related Architectural Scenarios

The business component infrastructure and client proxy framework mentioned in scenario #1 can be reused in scenario #3. The primary difference between the two scenarios is the implementation of the GUI and controller frameworks.

Scenario Assumptions

The following assumptions help frame the initial state of the application before the scenario begins:

- The schema defined by the ODS is fronted by basic business components that provide an object relational mapping of each table. Updates to the ODS are coordinated exclusively through the basic business components.
- Basic business components are defined as EJB Entity Beans or CORBA Business Objects served by the WebSphere Enterprise Edition application server Component Broker.
- This scenario is only focused on the flow of data between the Business Component Layer and the Legacy Data Domain.

Operational Flow of Scenario #3

This section defines the basic processing flow required to provide updates initiated in the Business Component Layer to legacy applications and the Legacy Data Domain (Figure 15).

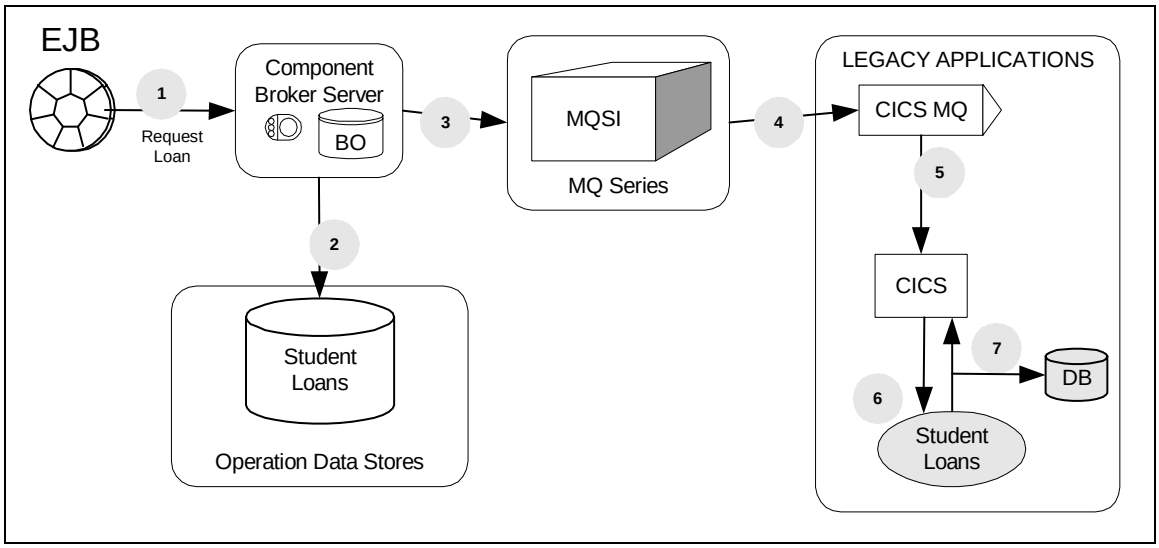

Figure 15 - Business Application Components Coordinating Legacy Updates

1. A Business Application Component (EJB Session Bean) issues an update to a Basic Business Component (EJB Entity Bean) that represents an atomic set of data stored in the Operational Data Store. The application object may be implemented using either of the component models, CORBA or EJB. The message is sent using IIOP (CORBA) or RMI over IIOP in the case of EJBs. In the case of internet applications a servlet is used as the controller to access the EJB or CORBA BO (CORBA Business Object).
2. The Basic Business Component provides an object relational mapping of the data it represents in the corresponding DB2 or Oracle database. The connection to the database is established using an application adapter defined by the container that is used by the business object server to coordinate qualities of services, such as transactional integrity. The Basic Business Components provide behavior that allows messages to be placed on a queue, to inform the Legacy Domain that an update has occurred. For example, a new *student loan* service has been implemented in using Department of Education's Enterprise Technical Architecture. Updates to the EDD *student loan* schema is coordinated directly by the business object server.
3. The Basic Business Component places a message on the queue that will notify the corresponding legacy application gateway that an update has occurred. In this case the Customer Information Control System (CICS) MQ Gateway is used to update a CICS CommArea with the data defined in the message. The EJB ensures that the message is placed on the queue by using the 2PC interface to the queue. Once on the queue, MQSeries guarantees that the message will be sent to the MQ server for routing to the proper application gateway. In this case the message is routed to the *Student Loan* gateway. The message destination is part of the information that the Basic Business Object places on the queue. MQSI provides the services for transforming the message data into the format required by the CICS legacy application. MQSI also routes the message to the correct location (machine) where the legacy *Student Loan* resides.

4. The MQSI pops the queue and reads the header information in order to determine where next to route the message and how the message may be transformed. MQSI maintains a set of rules that define how to format a message according to the target application. The message is then placed on the corresponding legacy application gateway queue for delivery. There may be several gateway interfaces available for MQSI to select that correspond to the same application. Thereby providing a level of load balancing and ensuring that the gateway is not a bottleneck.
5. Legacy gateways are implemented using a CICS MQ interface. The CICS application gateway pops the next message off the queue and coordinates the update to the corresponding application.
6. The target legacy application is responsible for making the proper updates to the Legacy Data Domain. Because of the intertwined nature of Department of Education's legacy applications and databases, only the legacy applications can successfully update the LDD.
7. The legacy application updates the corresponding databases with the information.

Benefits of Scenario #3

The primary benefit of implementing the architecture defined in scenario #3 is the ability to separate business logic from data logic. By having the Basic Business Object or entity object delegate the responsibility of updating the legacy applications, the legacy logic is effectively segregated from the Business Component Layer. If the associated legacy application is deprecated, the amount of impact on the Basic Business Component is minimal.

- Facilitates external reuse of enterprise components by platform specific implementation techniques. For example, use of MS-Visual Basic programming for Windows-NT.
- Provides a scaleable messaging infrastructure that can be used between enterprise services. For example, the *Loan Service* might communicate through MQSeries using MQSI to route and update the Claim service.
- Provides guaranteed delivery of messages between services and gateways.

Liabilities of Scenario #3

The liabilities of implementing applications that utilize scenario #3 are as follows:

- Long latency times between services could cause the Legacy Data Domain to be out of synchronization with the Enterprise Data Domain. Implementing load balancing and server replication can eliminate long latency times between services.
- Business Application Component response times may outstrip legacy application processing throughput resulting in application bottlenecks.
- Legacy application functionality must be thoroughly analyzed to properly implement legacy application gateways.

5.1.4. Scenario #4: Legacy Application Coordinating Business Application Components

Purpose

This scenario describes the process of coordinating business component updates from legacy applications. This scenario comes into play when business components have been developed to support the implementation of the EDD. However, the legacy application(s) that supports the corresponding LDD has not been deprecated because it still provides vital services to the business. Before a legacy application is deprecated it may be necessary for updates to legacy applications be propagated to the ODS (or other EDD databases). In this case updates to the ODS flow through the messaging layer and are routed to the proper application server using MQSeries Integrator.

In the near future legacy applications will maintain their external interfaces. This includes user and system interfaces. Eventually, business application components will be developed that mirror functionality provided by legacy applications. Over time legacy applications will be deprecated and replaced by new applications that support Department of Education's SFA Enterprise Technical Architecture. Until this occurs, updates posted to the Legacy Data Domain must be mirrored in the Enterprise Data Domain. Updates that must be propagated to the ODS in near real time and outside the ETL procedure use the process described by this scenario.

Architectural Pattern(s)

Scenario #4 is based the following architectural patterns:

- CORBA Services
- Enterprise Java Beans
- Java Applets
- Forward-Receiver (Provided by MQSeries)

Development Languages and Application Programming Interfaces

The following table defines the languages, APIs, and the purpose for using them within this scenario.

Table 9 - Scenario #4 Programming Languages and API's

Programming Language and APIs	Purpose(s)
Java	Develop business components. Develop interfaces to and from the business components to the MQ interface.
COBOL / CICS	Develop legacy application CICS interface to be used for implementing the external legacy gateway.

Programming Language and APIs	Purpose(s)
CORBA Event API	Leverage the CORBA event service to coordinate updates from business components to the MQ legacy interface.
MQI	MQ Application Programming Interface – used to implement the MQ server to route updates to the legacy domain.

Scenario Protocols

The following table lists the different protocols used to implement scenario #4.

Table 10 - Scenario #4 Protocols

Protocol	Purpose(s)
IIOP – Internet Inter-Orb Protocol	Provide network communication for CORBA and EJB objects. Develop business components
TCP / IP – Transmission Control Protocol / Internet Protocol	Network transport protocol used from client to web server
SNA Level 2 3270 Data Streams	Used by legacy applications to communicate in the OS / 390 legacy environment.

Related Architectural Scenarios

This scenario is closely related to Scenario #3 which defines the process for coordinating updates to the legacy data domain that originate from a business component in the application server. Scenario #4 is also closely related to Scenarios 5, 6, 7 and 8. Each of these scenarios use MQSeries Integrator to route updates to services throughout the enterprise.

Scenario Assumptions

The following assumptions help frame the initial state of the application before the scenario begins:

- The schema defined by the ODS (or other EDD databases) is fronted by basic business components (entity beans) that provide an object relational mapping of each table. Updates to the ODS are coordinated exclusively through the basic business components.
- This scenario is only focused on the flow of data between the LDD and the Business Component Layer.
- The legacy application gateway provides bi-directional support for transactions. Updates originating in the legacy domain can be passed to the gateway to update the Business Component Layer. Updates originating in the Business Component Layer can be passed to the legacy gateway for presentation to the corresponding legacy service.

Operational Flow of Scenario #4

This section defines the basic processing flow required to provide updates initiated by a legacy application to the Business Component Layer and the EDD (Figure 16).

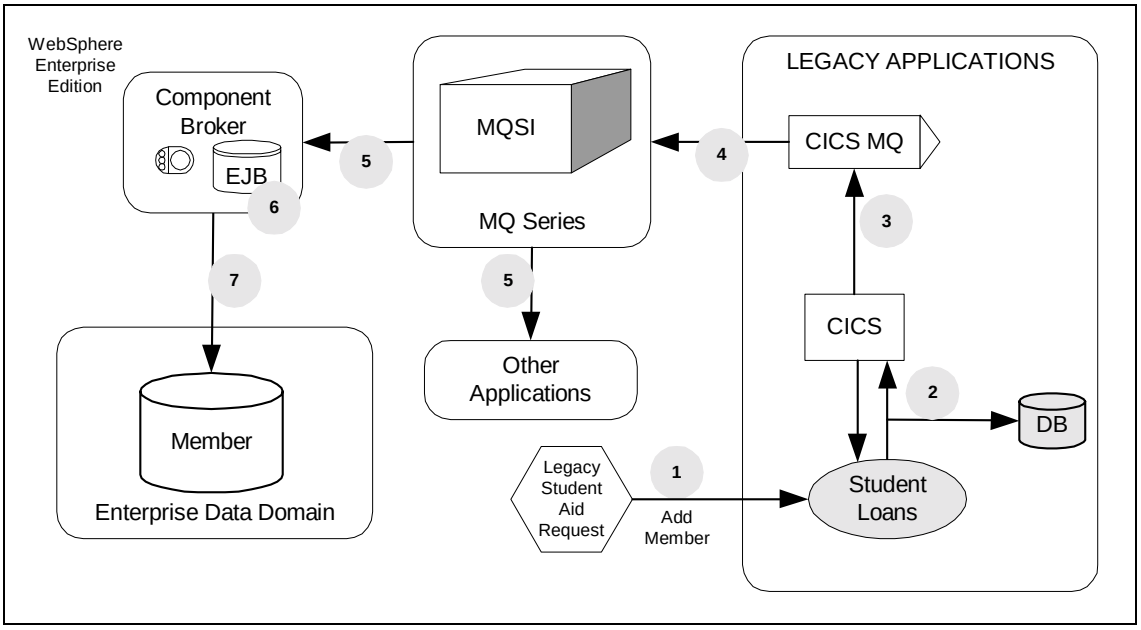

Figure 16 – Legacy Applications Coordinating Business Application Components

1. An update to the LDD is initiated through an interface to a legacy application. The update may originate from an end user or from an interface with another legacy service.
2. An interface to a legacy application initiates an update that results in a change to the LDD. This is the normal process for updating data by a legacy application in Department of Education's current environment.
3. The legacy application populates the CommArea that supports the legacy application gateway. This CommArea will be referred to as the Gateway CommArea (GCA). The legacy application is responsible for properly formatting the message supported by the GCA.
4. The legacy application then initiates the CICS application that supports the transfer of the message from the CICS CommArea to the legacy gateway application that is responsible for placing the message on the queue to be delivered to MQSI. The legacy gateway interface formats and transfers the information stored in the GCA to the queue responsible for routing messages to MQ Integrator. A legacy application may continue processing requests once a message has been placed in the GCA and the Gateway Transfer Program (GTP) has been successfully initiated.
5. The Application Interface Messaging (AIM) Server is notified of a message pending on the queue. The AIM Server pops the message from the queue and examines the header to determine the target destination. The message header informs the AIM Server that the message is intended for the Business Component Layer.

6. The Component Broker MQSeries Application Adapter (MQAA) is responsible for listening for messages sent from MQSI to the application server. The MQAA is notified of a message pending on the queue. The MQAA pops the message from the queue and examines the header to determine the business component for which the message is targeted. MQAA forks a new thread and sends the message to the business component responsible for updating the ODS or other data mart defined by the header of the message. An update to the ODS may require MQSI to formulate, transform and send multiple messages in order to update several back end systems or other applications. The MQAA provides a background process solely responsible for delegating messages sent from MQSI (or any MQ application) to the corresponding business component (EJB, CORBA BO).
7. The business component responsible for updates to the EDD defined by the message makes the requested change to the database. The two data domains are now synchronized. See Scenario #1 for more details on coordinating updates to the EDD using business components.

Benefits of Scenario #4

The principal benefit of scenario #4 is to provide access to Department of Education's legacy applications while supporting the implementation of the Enterprise Data Domain. Scenario #4 provides a process for supporting synchronized updates to both the Legacy Data Domain and the Enterprise Data Domain. Additional benefits include:

- Facilitates the gradual deprecation of legacy applications.
- Provides for reuse of established legacy user interfaces during transition to new services developed using the proposed technical architecture.
- Implementation of scenarios #3 and #4 will not require extensive modifications to legacy applications.
- Facilitates external reuse of enterprise components by platform specific implementation techniques. For example, use of MS-Visual Basic programming for Windows-NT.
- Provides a method of guaranteeing updates to the EDD initiated by legacy application.

Liabilities of Scenario #4

The liabilities of implementing applications that utilize scenario #4 are as follows:

- Directly updating legacy applications may circumvent business logic developed to support business application components.
- Long latency times between services could cause the Legacy Data Domain to be out of synchronization with the Enterprise Data Domain. Implementing load balancing and server replication can eliminate long latency times between services.
- Initial implementation of scenario #4 may negatively impact legacy application performance. A period of performance tuning may be necessary to properly integrate changes to the legacy environment that stem from the implementation of scenarios #3

and #4. Deprecation of existing legacy services will reduce the amount of message traffic flowing from legacy applications to the EDD.

- Legacy application functionality must be thoroughly analyzed to properly implement legacy application gateways.

5.1.5. Scenario #5: Using the Internet Portal to Search Content

Purpose

This scenario describes the process of finding information by searching for content using the enterprise portal. Content may consist of information located within or external to the enterprise. Information located within the enterprise may consist of data in data repositories, intranet URLs or documentation located in the document management system. Information located external to the enterprise consists solely of references to Internet URLs.

The Meta Group defines a portal as "A framework that enables differing levels of functionality (e.g. content, applications) and interactivity (e.g. community) to members based on preferences and business rules. Portals provide better "context" around work activities and add value to existing sites through customized connections." The Department of Education's SFA portal will provide the user community with a "gateway" to a set of services focused on financial aid for students. One such service will be the ability to search for information related to student financial aid. Users will access the portal via a well-known URL (e.g. **www.sfa.org**) and select the search option to display the search screen. The search screen will provide an input field in order to enter search criteria. Additionally, users will be able to select search parameters to aid in searching. Document references that match the desired search parameters will be categorized and displayed to the user on a search results screen. Users can view each search document by selecting the reference link provided by the search engine.

This scenario describes the technical architecture necessary to support searching through the SFA portal.

Architectural Pattern(s)

Scenario #5 is based the following architectural patterns:

- Content Searching
- Data Mining

Development Languages and Application Programming Interfaces

The following table defines the languages, APIs, and the purpose for using them with in this scenario.

Table 11 - Scenario #5 Programming Languages and API's

Programming Language and APIs	Purpose(s)
Java	Develop business components. Develop interfaces to and from the business components to the MQ interface.
Viador Portlet API	Used to provide access to customized references outside of the Viador environment.

Programming Language and APIs	Purpose(s)
Autonomy Search API	Used to search the Autonomy indexed repository.

Scenario Protocols

The following table lists the different protocols used to implement scenario #5.

Table 12 - Scenario #5 Protocols

Protocols	Purpose(s)
IIOP – Internet Inter-Orb Protocol	Provide network communication for CORBA and EJB objects. Develop business components
TCP / IP – Transmission Control Protocol / Internet Protocol	Network transport protocol used from the desktop application to the MQSI server.
RMI	Used to communicate between the Viador Portlet servlet and the VIC.

Related Architectural Scenarios

This scenario is related to the Data Propagation scenario. This scenario references data that is obtained as a result of the ETL process and Data Propagation scenario. Search results that reference internal enterprise data is maintained through propagation process.

Scenario Assumptions

The following assumptions help frame the initial state of the application before the scenario begins:

- The portal server and the search engine integration seamlessly.
- Searching is dependent upon the ETL and data propagation process and the availability of information stored in enterprise data warehouses and data marts.
- Accesses to external web references are dependent upon connectivity and availability of referenced sites.

Operational Flow of Scenario #5 – Searching Content using the SFA Portal

This section defines the process of using the enterprise portal to search for information. The following diagram displays the basic topology for supporting the implementation of Scenario #5. The portal server is supported by the Viador Information Center. Accesses to different portal functions are implemented using Viador Portlets. The search engine is supported through the use of Autonomy's Dynamic Reason Engine, HTTP Fetch Spider and AutoIndexer.

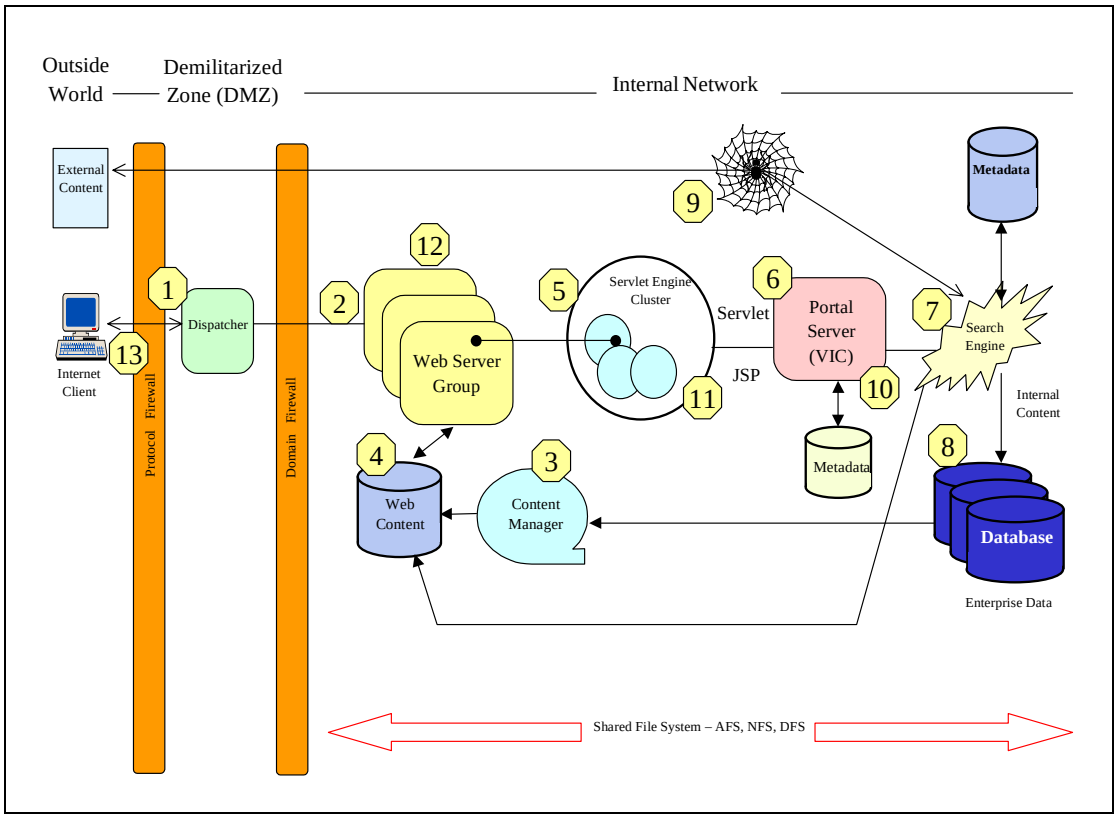

Figure 17 – Searching Content using the SFA Portal

1. A user accesses the SFA portal through their web browser via the Intranet or Internet. Figure 17 shows the topology necessary to support Intranet access to the public SFA portal. The user selects the search feature from one of the options within the portal.
2. The Web server is responsible for serving the web pages to the users web browser. Searching is supported through the web server using a reference to a servlet that accesses a Viador Portlet responsible for initiating the search. The integration between Viador and Autonomy is achieved using the Viador Portlet builder API to initiate search requests using the Autonomy search API.
3. Static web content is obtained from enterprise data sources and managed by the Content Manager (Interwoven TeamSite). This includes static web pages that are part of web applications and SFA documents.
4. Information served by the Web IBM HTTP Server (IHS) is stored in a web page repository on the file system.
5. The Viador search Portlet is accessed using a servlet that is referenced in the portal web page. The servlet / JSP engine provides support for execution of the servlet that access the Viador Portlet that is processed by the Viador Information Center (VIC).
6. The VIC processes the search request and passes the search parameters to the Autonomy search API.

7. The Autonomy Dynamic Reasoning Engine (DRE) searches the content repositories for information that matches the search criteria. This may references to Intranet and Internet URLs or references to documents stored in the document management system.
8. Enterprise content may consist of indexed data held in a database or documents that reside in the document management system.
9. References to Internet and Intranet information is obtained using the Autonomy HTTP Fetch Spider. Once information is obtained using the spider it is indexed by the Autonomy AutoIndexer to increase search speed.
10. References to search results are returned back from the Autonomy search engine to the Viador Information Center. Viador formulates the presentation of the search results using a JavaServer Page.
11. The JSP is compiled and the resulting HTML is returned to the IBM HTTP Server for presentation to the web client. The JSP includes the HTML page produced by the Autonomy search engine containing the requested search results.
12. The IBM HTTP Server posts the results of the search back to the User's web browser
13. Matching results can be viewed by selecting the document reference. The User selects the document reference in the result list for viewing. The referenced document is viewed through the web browser using one of the available browser viewer plug-ins.

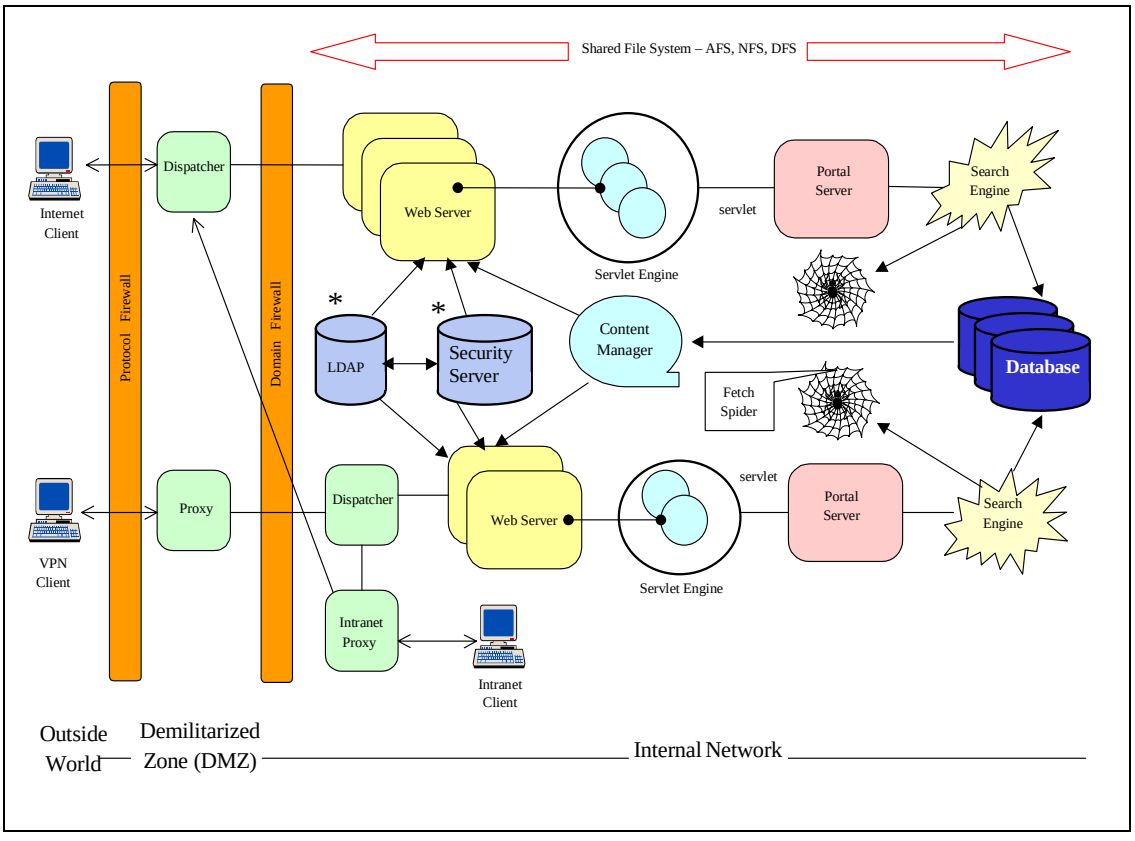

Figure 18 – Intranet and Internet Execution Topology

Operational Flow of Scenario #5 – Analyzing SFA Data Warehouses Through the SFA Portal

This section defines the process of using the enterprise portal to search and analyze data stored within the SFA Data Warehouses and Data Marts. The operational flow defined here follows the information shown in Figure 19.

1. A user accesses the SFA portal through their web browser via the Intranet or Internet. Diagram Figure 17 shows the topology necessary to support Intranet access to the public SFA portal. The user selects the option within the portal to search and analyze the SFA Data Warehouse. Figure 19 depicts the operational flow required for searching and reporting against SFA data stored in data warehouses. The Data Warehouse Portlet is invoked through a servlet that is defined within the search web page.
2. The Viador search Portlet is accessed using a servlet that is referenced in the portal web page. The VIC processes the search request and passes the search parameters to the MicroStrategy search API.
3. The MicroStrategy Intelligence Server processes the search or reporting request according to the parameters passed to the search engine via the Viador Portlet.
4. The data that matches the search or report request is formatted into an HTML page using the result data that matches the search criteria.
5. Customized Portlets can be developed that combine search results from both data warehouse reports and content search results obtained from the Autonomy search engine.
6. SFA Data Warehouse reports and search results are referenced as URLs and presented to the client as HTML through the WebSphere servers.

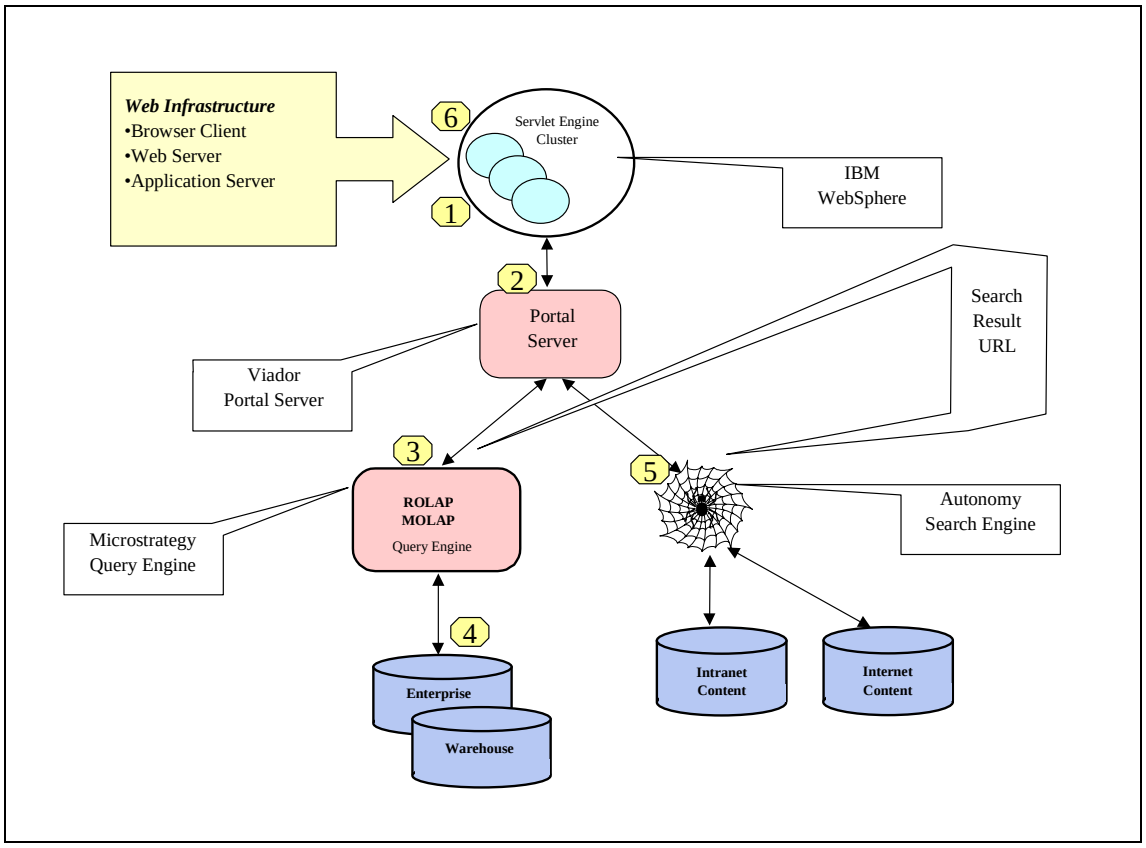

Figure 19 – Analyzing Data Warehouses using the SFA Portal

Benefits of Scenario #5

- Provide the end user with a consistent look and feel through out the SFA web sites.
- Integration between the Viador Portal product and the Autonomy Search product will provide SFA Users with a well integrated enterprise search tool.

Liabilities of Scenario #5

- Developers will need to implement custom Portlets to provide seamless integration between Viador and Autonomy.
- Additional interfaces between Viador and the search engine may impact performance and will require operational oversight and administration.

5.1.6. Scenario #6: Data Population Using an ETL Process

Purpose

This scenario describes the process and operational flow of populating data warehouses, data marts and operational data stored within the Enterprise Data Domain.

Data Population Strategy

Few, if any, data warehouse population processes are simple enough to be performed in a single step. Therefore the basic 'building block' which works on the (read dataset – process – write dataset) model is then applied to the problem in as many steps as the designer thinks is appropriate for the problem being solved.

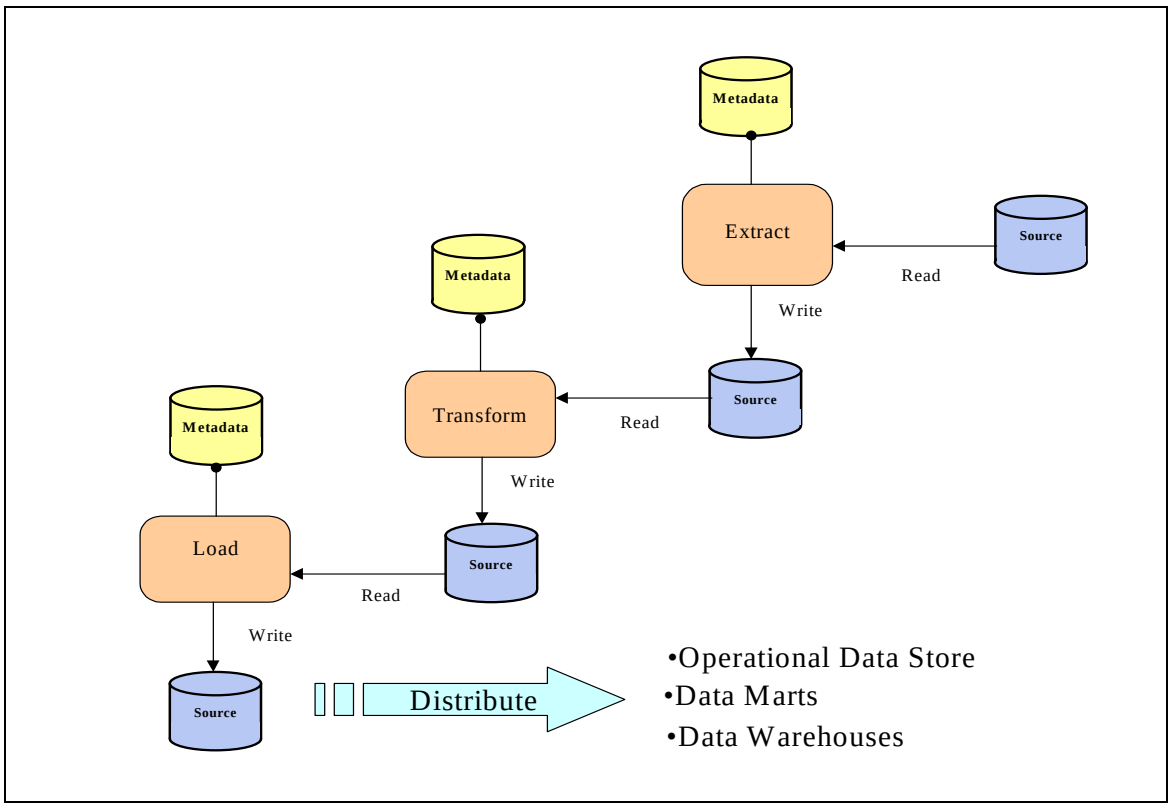

Figure 20 – The Extract, Transform and Load Process

Extract, Transform and Load Overview

A process known as Extract, Transform and Load (ETL) is used to populate data stores from source data sets in the EDD. The ETL process is supported through the use of Informatica PowerCenter. The SFA source data is the data that resides in the databases of all the legacy applications, the LDD. Data that resides in the databases of the LDD is considered 'dirty.' Data is 'dirty' when information is duplicated, fragmented, not normalized and insufficiently

categorized. Data is considered 'clean' once it has been extracted from the legacy repositories and transformed into data that adheres to a specific data model that is more useful to the enterprise.

The ETL process (above) is used to populate three different types of repositories, Data Marts, Data Warehouses and the ODS. Data Marts are repositories that support different segments of the business that require diverse data models. Data Warehouses are large repositories used as clearinghouses for data used as input for analytical processing. The ODS is a repository used to support the data requirements necessary to support the daily operation of strategic enterprise applications. The vast majority of the ODS is considered 'read-only' data that is repopulated nightly from the LDD by the ETL process. However, this strategy may change as new applications are introduced into the enterprise and older legacy applications are deprecated. As a result the ODS may be considered the repository that reflects the enterprise data model. Additional information can be found on this topic in the Data Population Scenario.

Extract Process

The extract application extracts data from the source dataset, in the case of SFA, the LDD. The dataset is owned by one of the existing legacy applications within SFA. More than one legacy repository may be involved in the extraction process. Data is obtained from legacy repositories using a set of extraction rules. Extraction rules define the specific data elements that are copied from the source and the format of the data when written to the target database.

Transformation Process

The transformation application converts source data from one form to another according to a set of supplied rules. Transformation includes a variety of operations that can be performed on the source data. This includes joins, validation, data cleansing, data replacement, deletion or any other operation defined by the transformation rules. The source data for the transformation process is the output from the extraction process. Transformation is often a non-trivial process because generally there is no guarantee that all fields will be present when required, and the transform process must be able to handle this condition.

Load Process

The Load process uses the output of the Transformation process to load the enterprise data repositories. This includes enterprise data warehouses, data marts and operational data stores. The load process may remove any existing data from target databases or simply update data already present. For example, it is more efficient to replace data in operational data stores and data warehouses. However, it is more efficient (and safer) to update tuples in a data mart repository

Architectural Pattern(s)

Scenario #6 is based the following architectural patterns:

- Data Warehouse Management
- Extract, Transform, Load and Distribute Pattern

Development Languages and Application Programming Interfaces

The following table defines the languages, APIs, and the purpose for using them with in this scenario.

Table 13 - Scenario #6 Programming Languages and API's

Programming Language and APIs	Purpose(s)
ODBC	Database connectivity
Native Application Adapters	Supported through Informatica to support database connectivity
HTML	Support connectivity to administrative ETL interfaces Provide support for implementation of OLAP reports.
XML	Provide a common message format for data exchange and flexible message communication.

Scenario Protocols

The following table lists the different protocols used to implement scenario #5.

Table 14 - Scenario #6 Protocols

Protocols	Purpose(s)
TCP / IP – Transmission Control Protocol / Internet Protocol	Network transport protocol used from the desktop application to the MQSI server.

Related Architectural Scenarios

This scenario is related to the Data Propagation scenario. This scenario references data that is obtained as a result of the ETL process and Data Propagation scenario. Search results that reference internal enterprise data is maintained through propagation process.

Scenario Assumptions

The following assumptions help frame the initial state of the application before the scenario begins:

- Stateful business components (EJBs, CORBA BOs) that support OLTP utilize the ODS to provide persistence to entity objects.

Operational Flow of Scenario #6

This section defines the basic steps necessary to propagate data contained in legacy databases to Data Warehouses, Data Marts and ODS within the SFA Enterprise. The following steps follows the operational flow depicted in Figure 21.

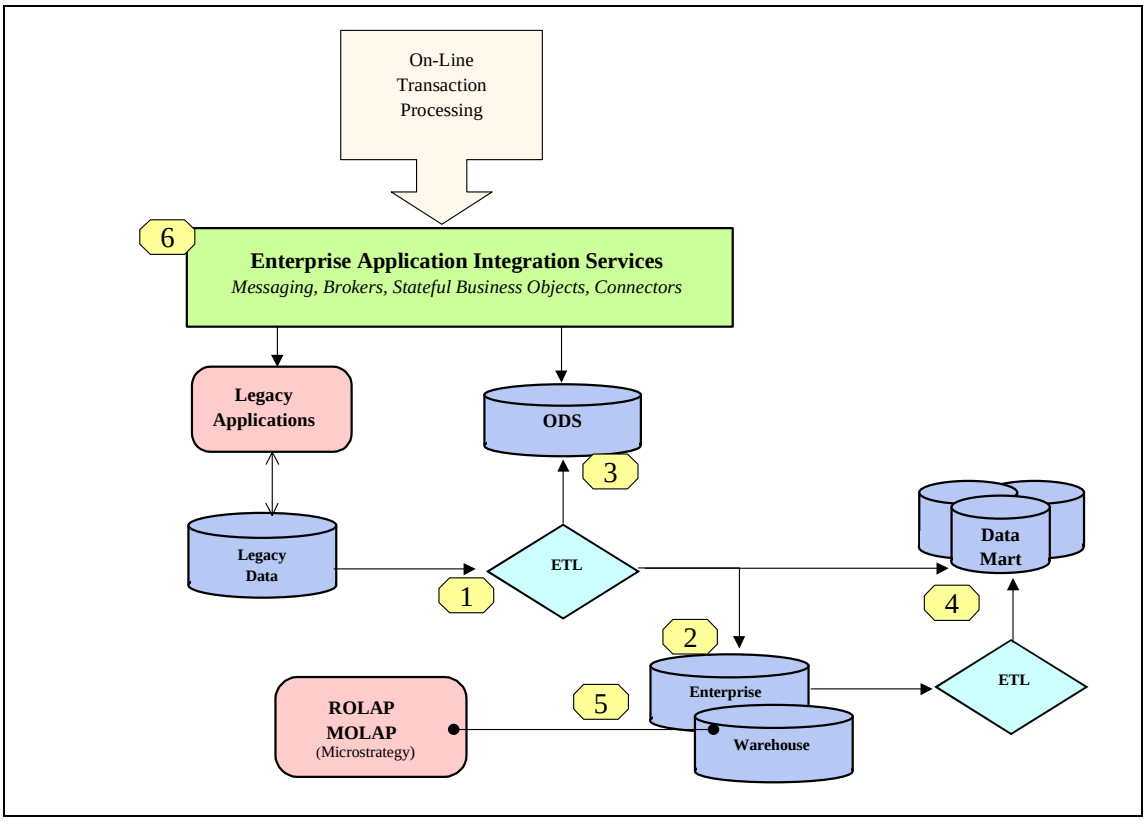

Figure 21 – Data Propagation within SFA using an ETL Process

1. Data stored in the LDD is extracted according to data type and purpose, transformed according to SFA schema models, and loaded into target Data Warehouses, Data Marts or ODS. The process is accomplished using the Informatica PowerCenter ETL server.
2. Information that is extracted from Legacy Data repositories is placed into the SFA Data Warehouse. The Data Warehouse schema supports the end-user access via ad-hoc queries, analytical processing and Knowledge Discovery using intelligent mining tools.
3. A separate ETL process extracts data from multiple legacy data repositories to build the ODS. The ODS schema is more normalized than the Data Warehouse in order to support applications that implement stateful business objects that utilize the ODS for persistence.
4. Data may undergo additional ETL conversions to be distributed to Data Marts. The schema of each Data Mart reflects the requirements of the corresponding user community. Alternatively, a Data Mart ETL source may originate from ODS.
5. On-line Data Mining is supported through the use of the MicroStrategy Intelligence Server and other reporting agents.
6. Data stored in Legacy applications are updated through the EAI Domain. When necessary data contained in the ODS is updated directly by stateful business objects using application adapters or connectors provided by the EAI.

Benefits of Scenario #6

- Provides the processes and services that move and control movement of data, resulting in the population of the enterprise data stores.
- Supports end-user access to data stored within the SFA Data Warehouses.
- Provides the mechanisms and architecture to access and display data in an understandable and flexible manner.
- The implementation of ODS defined by an Enterprise Data Schema better supports the SFA business.

Liabilities of Scenario #6

- ETL processes are often resource intensive operations. The appropriate level of resources must be allocated to each step of the ETL process.
- Data stored in within the LDD can become out of sync with information stored in ODS.

5.1.7. Scenario #7: Coordinating Transaction Processing

Purpose

This scenario describes the process of coordinating transactions between services and domains within Department of Education's ETA. There are three different domains. The Enterprise Domain includes the Business Object Server, EDD and any supporting application services. The Legacy Domain includes the LDD and the supporting legacy applications and services. The Back Office Domain includes MS-Windows based applications that depend upon business services provided by the Enterprise Domain to support their processing.

Transactional integrity must be maintained within and between services, domains and back end resources in order to protect data integrity and guarantee robustness. Interfaces between services provided by Component Broker and the MQSI servers must provide a transactional process for ensuring the delivery messages between the servers. Transactional integrity must also be maintained between distributed services that utilize the same back end resources. For example, Application Business Components must have the ability to coordinate updates to multiple databases in the EDD within a single transaction (Figure 22).

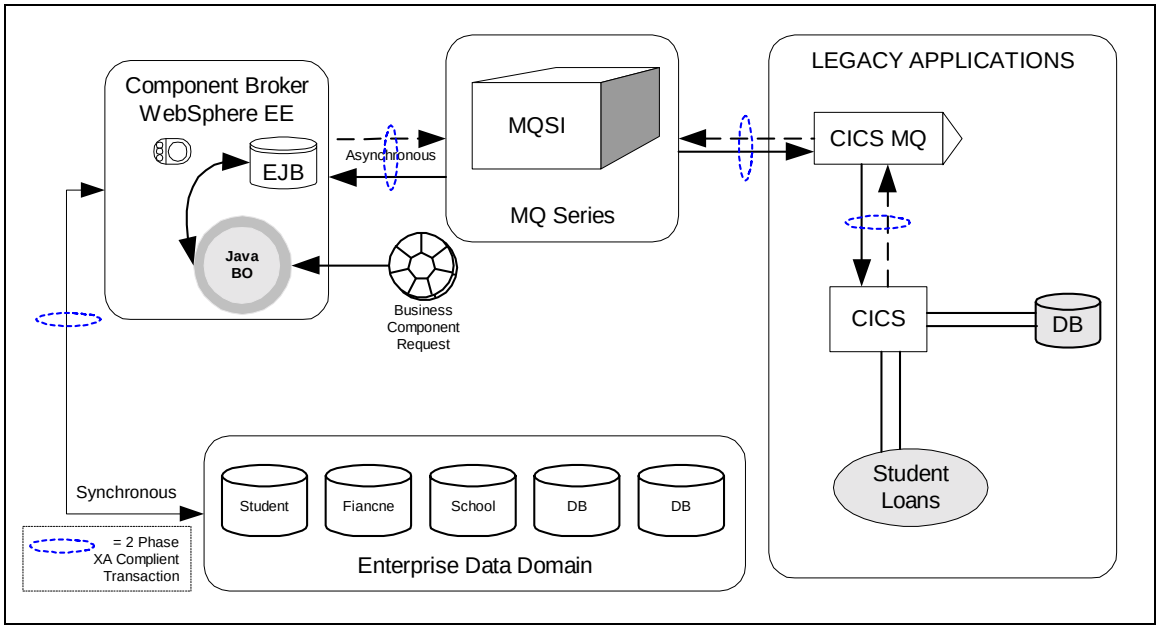

Figure 22 - Coordinating Transactions within the Enterprise

The process of coordinating distributed transactions must also include an exception flow should a transaction abnormally terminate. This is especially true for message based updates between the Enterprise and Legacy Domains. There must be a guaranteed method of ensuring that once a message has been sent that it gets delivered to the target.

Architectural Pattern(s)

Scenario #7 is based upon the following architectural patterns:

- CORBA Services
- Enterprise Java Bean Transaction Specification (OTS)
- X/Open 2 Phase Transaction Specification
- Forward-Receiver (Provided by MQSeries)

Development Languages and Application Programming Interfaces

The following table defines the languages, APIs, and the purpose for using them within this scenario.

Table 15 - Scenario #6 Programming Languages and API's

Programming Language and APIs	Purpose(s)
COBOL/CICS	Develop legacy application CICS interface to be used for implementing the external legacy gateway. Provide transactional integrity for OS/390 CICS based applications.
CORBA Event API	Leverage the CORBA event service to coordinate updates from business components to the MQ legacy interface.
CORBA Transaction Service	Enables distributed work to be conducted in a coordinated fashion.
EJB Object Transaction Service	Defines interfaces and semantics for EJB based transaction service.
Java	Develop business components. Develop interfaces to and from the business components to the MQ interface.

Scenario Protocols

The following table lists the different protocols used to implement scenario #7.

Table 16 - Scenario #7 Protocols

Protocols	Purpose(s)
IIOP – Internet Inter-Orb Protocol	Provide network communication for CORBA and EJB objects. Develop business components
TCP/IP – Transmission Control Protocol / Internet Protocol	Network transport protocol used from the desktop application to the MQSI server.
APPC – SNA LU 6.2	Network transport protocol used for SNA based client server applications. Provides 2PC for SNA based applications.

Related Architectural Scenarios

All previously defined scenarios depend upon the process defined here to coordinate transactions.

Scenario Assumptions

The following assumptions help frame the initial state of the application before the scenario begins:

- The example cited in this scenario is focused on coordinating transactions between the EDD, the Business Object Server, and the Message Application Routing Service (MQSI).
- The schema defined by the EDD is fronted by Basic Business Components that provide an object relational mapping of each table. Updates to the EDD are coordinated exclusively through the basic business components. It is the responsibility of Application Business Components to coordinate transactions with Basic Business Components. Transactions may be coordinated either programmatically or directly through the container that manages the component. The exact method depends upon the vendor implementation of the Business Object Server.

Operational Flow of Scenario #7 – Coordinating Transactions: Normal Flow

This section defines the conceptual operational flow for coordinating transactions between resources within Department of Education's Enterprise Architecture. The process defined in this section describes the normal flow for message transactions between the Business Object Server and the Message Application Routing Service as shown in *Figure 23 - Coordinating Transactions between the MQSI and Business Object Servers.*

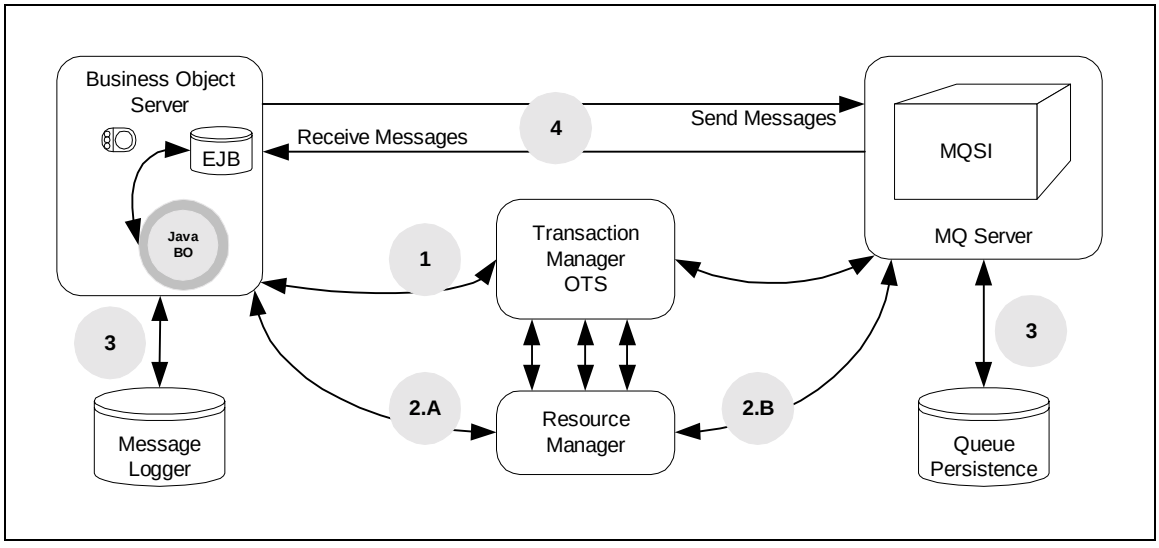

Figure 23 - Coordinating Transactions between the MQSI and Business Object Servers.

1. Upon start-up of Component Broker Application Server an instance of the Transaction Manager is initialized. Most Business Application Servers based on the CORBA or EJB

specifications use the Object Transaction Service (OTS). The OTS provides for the implementation of transactional objects and servers.

2. The Business Object Server (2.A) and the MQ Server (2.B) register their resources with the OTS once a request for their resources has been initiated. Some OTS vendor implementations allow for static registration with the OTS upon start-up of the resource. Either way the Business Object Server and the MQ Server register their resources. The resources establish the nature of the transaction and their attributes.

   A. The Business Object Server checks the Message Logger for any pending messages that need processing. The Message Logger contains messages that were unable to be processed since the server was shutdown. If a message exists in the Message Logger, an error occurred while trying to forward a message to the MQSI server. If unprocessed messages are being held in the Message Logger, they are processed before any new messages are sent to MQSI.

   B. The MQ Server checks the dead letter queue and processes any messages that could not be delivered during the last cycle of the server. Messages that cannot be delivered are sent to the dead letter queue and stored in long term Queue Persistence. The processing of dead letters (undeliverable messages) is defined administratively within MQSeries by establishing a set of rules for dealing with undeliverable messages.

3. Messages sent by MQSI to a Component Broker server using the MQAA are placed on the queue within a transactional context. Messages placed on a queue by the MQSI server are coordinated with the MQSeries Transaction Manager using a 2-phase commit. The MQSI server is notified of any failure to place a message on a target queue.

4. Messages sent by Component Broker to the MQSI server via an MQSeries queue are transactionally coordinated through the CB OTS and the MQ Application Adapter. Updates to the ODS are coordinated by Basic Business Components (entity objects – EJBs or CORBA BOs) that utilize an application adapter to implement the actual update to the database. Multiple database updates may be coordinated through a single synchronous transaction. However, the same update may require several messages to update the Legacy Data Domain with the same information. These messages are placed on the queue to be sent to the MQSI server within one transactional context. This ensures that either the whole update is sent to the LDD or none. An alternative to this method is to list all related messages in the message header. Once the legacy gateway receives all related messages, they are processed for update to the LDD. It is suggested that both these precautions be taken when developing the MQSI server Figure 24 shows how a single update initiated from an Application Business Component can translate into multiple messages to be sent to the MQSI server.

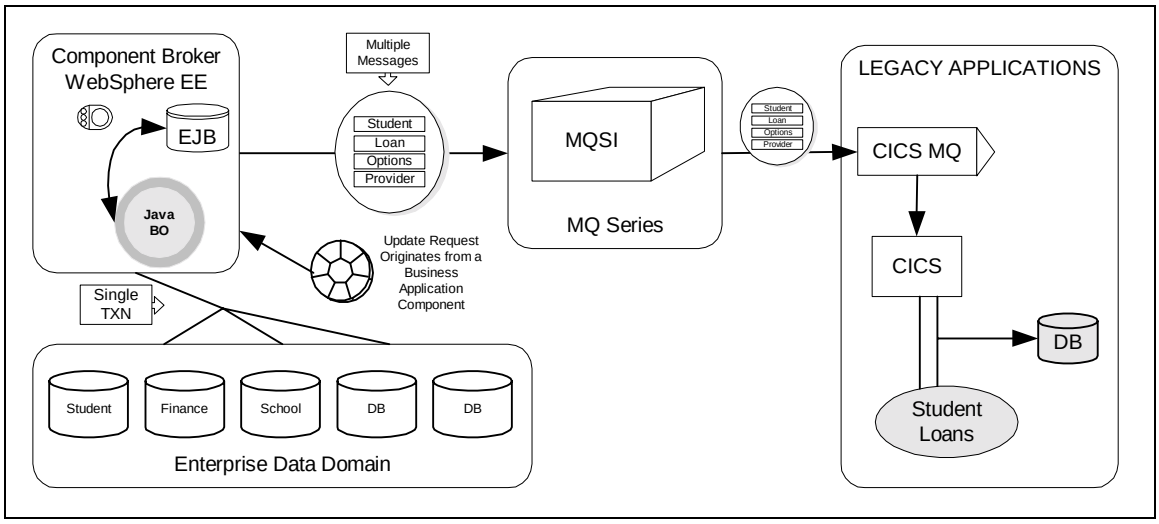

Figure 24 - Single EDD Transactions Translate into Multiple Messages

Operational Flow of Scenario #7 – Coordinating Transactions: Exception Flows

This section defines the conceptual operational flow for processing transactions that terminate abnormally. This operational flow is focused on transactions that occur between the WebSphere Business Object Server, Component Broker and the MQSeries Integrator. There are two major exception flows that can occur between the CB and MQSI servers. An abnormal termination occurs when one of the servers is unable to place a message on the queue used to transport messages between the servers. One exception flow occurs when the MQAA home object cannot place a message on the queue for delivery to the MQSI server. The other exception flow occurs when the MQSI server cannot place a message on the queue for delivery to the CB server.

Multiple updates to the EDD can be coordinated using a single transaction. This is a synchronous update to the EDD. However, multiple messages may be necessary in order to communicate these updates to the LDD. Therefore, a single EDD transaction may result in one or more Service Messages that need to be routed through the MQSI server to the legacy or back office domain. This type of update to the legacy or back office domain is asynchronous because it is broken down into parts and transported via the queue. The process for handling exceptions that occur during asynchronous updates is different from synchronous updates.

The processes defined by these operational flows are applicable for handling exceptions during transactions among other processes within the architecture.

Exception Flow for Transactions that Originate from the MQSI Server

The process defined in this section describes the exception flow that occurs when the MQSI server is unable to place a message on the queue for delivery to the CB server (Figure 25).

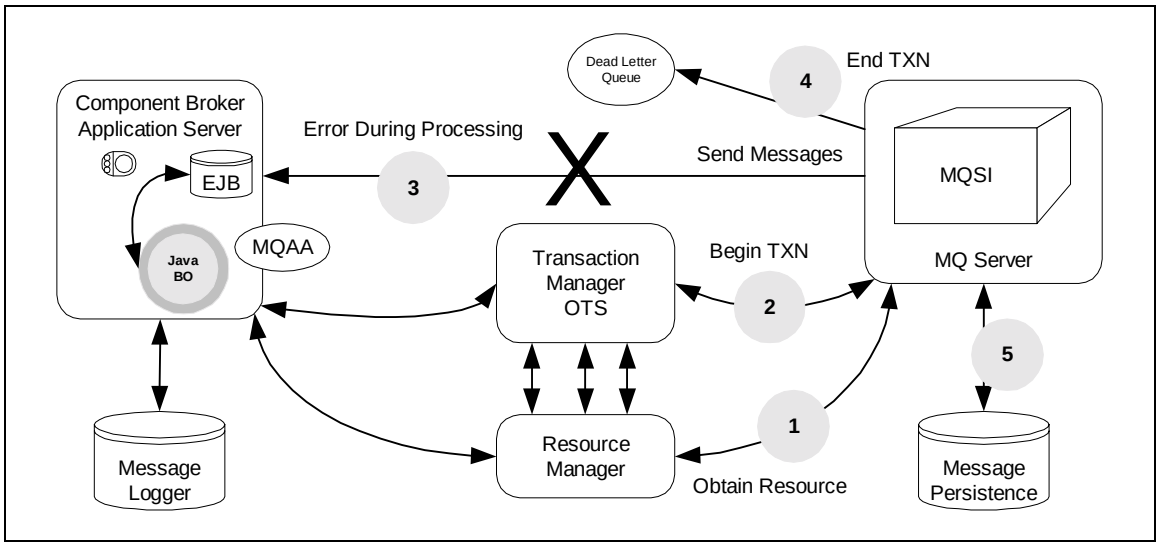

Figure 25- Exception Flow for Transactions that Originate from the MQSI Server

1. The MQSI server obtains the resources for sending a message to the CB server. In this case the resource is a reference to a queue between the MQSI server and the CB server. There may be several queues between services. Multiple queues improve performance through workload balancing. The Resource Manager selects the queue with the least load.
2. The MQSI server initiates a transaction with the Transaction Manager that will be used to coordinate the placement of messages onto the queue bound for the CB server. There may be several messages that are part of the same transaction. The Transaction Manager coordinates the placement of all the messages onto a queue that are part of the same transaction. If any message fails to be placed on a queue then the previously placed messages are rolled-back from the queue.
3. The MQSeries Transaction Manager coordinates the placement of the messages onto the queue bound for the CB server. However, during the 2-Phase commit process an exception occurs. For example, the queue crashes due to a hardware failure and is no longer accessible. The transaction never completes. The CB server can perform one of the following three options to rectify this situation.
4. Retry the transaction again and place the messages on another queue as specified by the Transaction Manager.
5. If the transaction cannot be successfully completed then the MQSI server can place the messages into a dead letter queue for later processing.
6. However, if a catastrophic MQSeries failure occurs then the dead letter queue may not be available. In this case the MQSI server will temporarily store the messages in persistent storage for future processing. The MQSI servers will then shutdown and alert the console of the problems encountered. Once restarted the MQSI server will reprocess any messages found in the dead letter queue or temporary message persistence.

Exception Flow for Transactions that Originate from the CB Server

The process defined in this section describes the exception flow that occurs when the CB server is unable to place a message on the queue for delivery to the MQSI server.

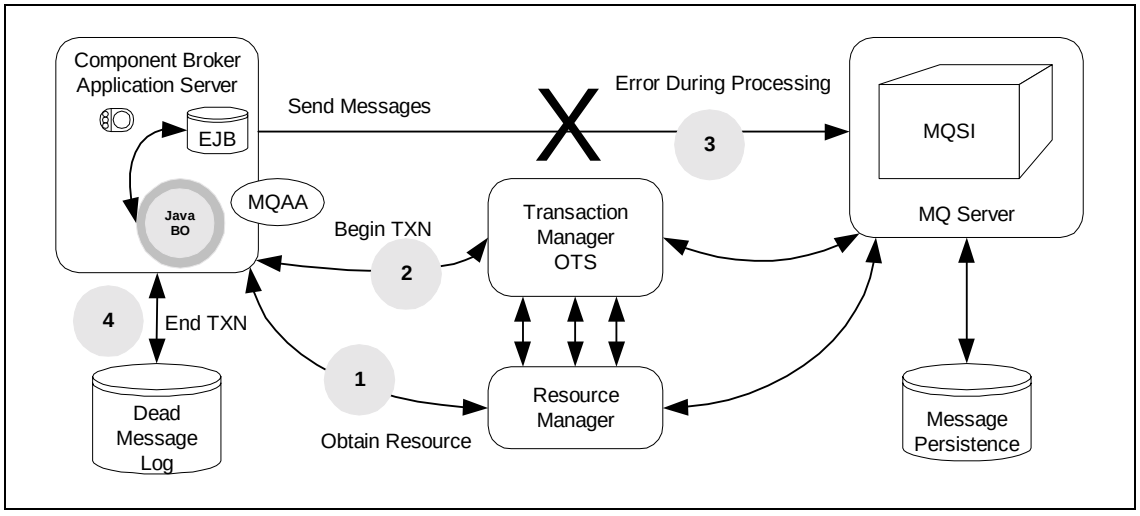

Figure 26 - Exception Flow for Transactions that Originate from the CB Server

1. The CB server obtains the resources for sending a message to the MQSI server. In this case the resource is a reference to a queue between the CB and MQSI servers. There may be several queues between services. Multiple queues improve performance through workload balancing. The Resource Manager selects the queue with the least load. This process occurs under the hood using the CosTransaction service or the EJB OTS.
2. The Object Transaction Service [Transaction Manager] sets the scope of the transaction. The CB server initiates a transaction with the OTS that will be used to coordinate the placement of messages onto the queue bound for the MQSI server. There may be several messages that are part of the same transaction. The Transaction Manager coordinates the placement of all the messages onto a queue that are part of the same transaction. If any message fails to be placed on a queue then the previously placed messages are rolled-back from the queue. This process is only initiated after successful completion of the transaction responsible for coordinating the update to the EDD with the same data.
3. The OTS coordinates the placement of the messages onto the queue bound for the MQSI server. However, during the 2-Phase commit process an exception occurs and the transaction aborts. For example, the queue crashes due to a hardware failure and is no longer accessible. The CB server can perform one of the following options to rectify this situation.

   A. Retry the transaction again and place the messages on another queue as specified by the Transaction Manager.

   B. However, if a catastrophic MQSeries failure occurs then CB will be unable to send messages via the queue through the MQAA. In this case the CB server can temporarily store the messages in a file for persistent storage for future processing.

This file is called the Dead Message Log (Figure 26). An interval can be established for CB to examine the Dead Message Log. If the Dead Message Log is found to contain messages then another attempt is made to send them to the MQSI server via the queue.

Benefits of Scenario #7

The principal benefit of implementing the architecture described by scenario #7 is the ability to coordinate distributed transactions and ensure the proper use of system resources. Coordinating updates to back end resources using transactions guarantees database integrity and message delivery. Other benefits of using transactions to coordinate updates with Department of Education's Enterprise Architecture are as follows:

- Transaction Managers make efficient use of system resources and facilitate the implementation and administration of work-load management.
- Provides a process for safely managing system exceptions and facilitates the recovery from system failures.
- Guarantees delivery of messages between system services that use MQSeries and the MQSI server.

Liabilities of Scenario #7

The liabilities of implementing the architecture described in scenario #7 are as follows:

- Proper analysis and design of a transactional system is required in order to avoid potential resource concurrency problems. Resources may become unavailable to other processes if locked as the result of long lasting transactions.
- Performance testing must be implemented using transactional entities in order to determine system performance requirements. Transactional systems often require additional processing overhead.

5.1.8. Scenario #8: Accessing Applications Through The DMZ

Purpose

This scenario defines a secured Internet based thin client architecture. The goal of this architecture is to provide the security aspect of the Scenario #1 – how to handle authentication, authorization, and message protection in a distributed object environment.

Architectural Pattern(s)

Scenario #8 is based the following architectural patterns:

- Model View Controller
- CORBA Services
- Enterprise Java Beans
- Web Servlets
- Java Server Pages (JSP)

Development Languages

The following table defines the languages and the purpose for using them with in this scenario.

Table 17 Scenario #8 Programming Languages and APIs

Programming Languages and APIs	Purpose(s)
Hypertext Markup Language – HTML	Develop the presentation of the web page. Define the location of the web server. Execute servlets on the server
Java	Develop business components Develop servlets Develop JSPs
C++	Develop CORBA business components invoked by other business objects such as CORBA JavaBOs and EJBs.

Scenario Protocols

The following table lists the different protocols used to implement scenario #8.

Table 18 - Scenario #8 Protocols

Protocols	Purpose(s)
HTTPS – Hyper-Text Transport Protocol	Used to send secure information to and from the web client and web server. Provide encryption of html to and from browser and web server.
HTTP – Hyper-Text Transport Protocol	Used to send information to and from the web client and web server.
SSL – Secure Sockets Layer Protocol	Used to provide authentication and message protection.
PKI – Public Key Infrastructure	Provide a framework that uses a pair of keys in authenticating a principal.
IIOP – Internet Inter-Orb Protocol	Provide network communication for CORBA and EJB objects.
TCP / IP – Transmission Control Protocol / Internet Protocol	Network transport protocol used from client to web server

Related Architectural Scenarios

This scenario is closely related to the thin client architectural scenario. The thin client scenario outlines an architectural pattern used to define a Internet access to enterprise resources. This scenario assumes security access from the Internet client to the web server and between the web server and application server.

Scenario Assumptions

The following is a list of assumptions, which help to define the state of the system before the scenario begins:

- **An application that uses HTML served by a web server. This server is capable of executing servlets and Java Server Pages.**
- **Secured Access to the web server and application server.**

Operational Flow of Scenario #8

This section defines the basic processing flow required to provide thin client access to business application components.

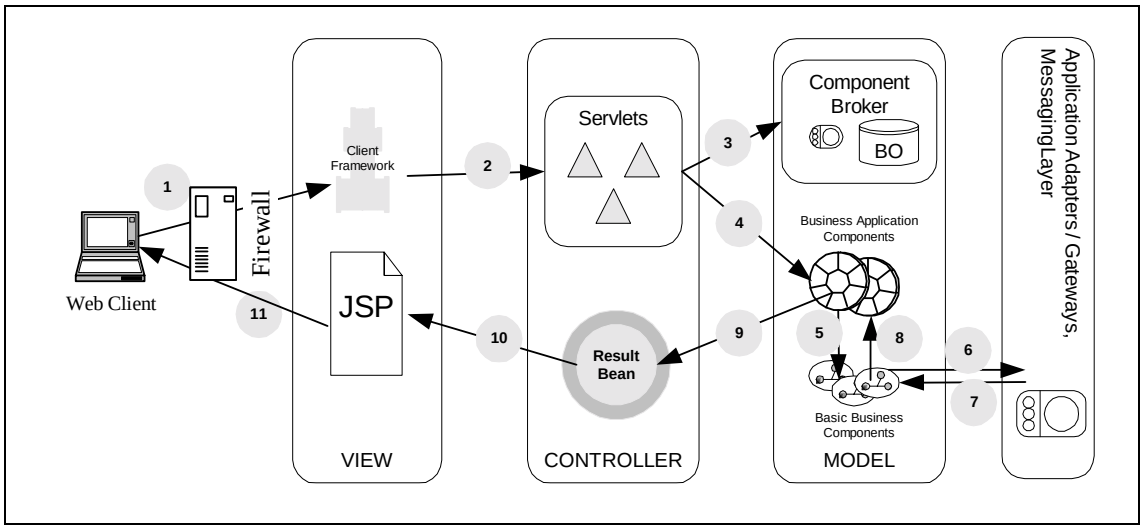

Figure 27 - Scenario #8 (Accessing Applications Through The DMZ)

The processing flow is the same as the one documented in Scenario #1. The following discussion is focusing on the security flow where needed. Security measures are set up at the application development and system configuration time. Most of the measures are transparent to the end user and application operational flow.

- Arrow #1. There is a security set up process before the first web page is presented to the user. A secure connection is established between the Browser and the Web Server using Secure Socket Layer (SSL) protocol and Public Key Infrastructure (PKI). When the Browser first presents a request to the Web Server, the Web Server sends back to its certificate (server-certificate, issued by a Certificate Authority (CA)). The Browser decrypts the server-certificate with CA's public key and authenticates the certificate itself. To complete the server authentication, the Browser sends a message encrypted with the Web Server's public key (part of the server-certificate) to the Web Server. The Web Server decrypts the message with its private key and sends the original message along with the answer back to the Browser. The Browser decrypts the message with the Web Server's public key. If the answer and message are satisfactory, the Web Server authenticity is established. At this time, the Browser creates a session key based on the information related to the session (e.g., date/time) and encrypts it with the Web Server's public key and sends it to the Web Server. Both the Browser and the Web Server then use the session key to encrypt and decrypt subsequent messages during the session and thereby, establish a secure communication channel between them. The setup process seems lengthy, but it happens 'under the cover'. The user and application are not aware of and are not involved in it.

Using the secure channel, the Web Server can further authenticate the Browser (client) by requesting from the Browser a user ID and password. The user ID and password is then verified against the user registry.

- Arrow #3. The controller servlet and the application server must pass the authentication verification upon the first method request against the application server. There are a couple of ways for this process – 2-party and 3-party authentication. 2-party

authentication uses the PKI settings and the same SSL protocol as described above between the Browser and the Web Server. Once authentication is satisfied, the user ID / password can be passed to the application server using the secure channel.

3rd-party authentication involves the servlet, the application server and a trusted third party – a security server. The servlet and the application server each authenticate itself to the security server and each gets back a secure token, representing its authenticity. In the servlet case, the user's identity (user ID / password) is used when authenticating to the security server. In the application server case, its identity is created and protected at installation and configuration. The application server authenticates to the security server when it is started. The tokens are passed between the servlet and the application server and the receiver can verify it with the security server.

The application server uses the user ID / password to form a credential representing the authenticity of the client (the user). This credential is used to enable any work initiated at the application server under the client authority – access controls are enforced based on the client privilege attributes.

- Arrows #4 & #5. If the server-level authorization is enabled at the application server, the client's authority to execute operations in the application server is based on the privilege attributes assigned to the client's credential and the Access Control Lists (ACLs) specified in the server-level security domain manager. If the client is granted the *operation:execute* right, the requested operation is forwarded on to the business component for execution.

If, on the other hand, the method-level authorization is enabled at the application server, the required-rights for the invoked operation is compared to the client's granted rights based on the privilege attributes assigned to the client's credential and ACLs specified in the object-level security domain manager. If they compare positively, the requested operation is forwarded on to the business component for execution.

There could be occasions where the logic in the business component is conditioned on the client's privileges. For example, the business logic wants to prevent a claim adjuster from processing his / her own claims. To accomplish this, the *adjustClaim* method logic needs to know who invoked the method.

In the implementation of the method, it first needs to get a security Current object. From the Current object, it then gets the client's Credential object. Finally, from the Credential object, it gets the identity type of the privilege attribute.

- Arrow #6. The application adapter needs to form a connection with the specific database before issuing requests to the database to obtain data for the state of the basic business components. Assuming the installation configuration uses the authorization policies in the data system, the application adapter retrieves the mapped security information for the specific database from a sign-on server and logs the client in to the database as part of connecting to it.

Benefits of Scenario #8

The benefit of scenario #8 is that it adds the security mechanisms to the distributed object environment. It provides authentication of principals (client and server), access control, and

message protection. This greatly reduces vulnerabilities and exposures that are inherently existed in a heterogeneous distributed systems. Additional benefits are:

- Provides a separate sign-on server that maintains legacy system login information. Each legacy system has its own access control mechanism and there are many systems in the existing environments (e.g., DB2, CICS, IMS, SAP). The sign-on server provides a single and safe repository for the login information. The users have to only remember one user ID / password and the application sever will retrieve the appropriate login information to connect to various legacy data systems on his / her behalf.
- The 2nd firewall provides an extra layer of security to the enterprise system.
- The security mechanism follows the industry standard where they exist (e.g., CORBA security service).
- Makes implementation of the DMZ architectural pattern easier. Separation of the web and application servers restricts the flow of IIOP and other enterprise specific network protocols (such as SNA) to with in the trusted network.
- No direct access to business components thereby adding a degree of security.
- Promotes load balancing and security of the physical layers – each server (web, application, security, sign-on, database) is on a separate machine. Each server can also be replicated for scalability.

Liabilities of Scenario #8

The liabilities of implementing applications that utilize scenario #8 are as follows:

- The security mechanism adds a lot of hand-shakes and checks in the processing. Depending on the level of authorization enabled, it could add a fair amount of cycles to a given method invocation.
- Separate organization must be formed to define standards for rights labels and rights family for use in the access control. Each development staff must understand how and when to use the correct right labels when developing method specifications.
- Separate organization must be formed to maintain the user registry. It includes user ID, password and privilege attributes. The registry is the repository that provides key information for authorization and access control.

6 Conclusion

The technical architecture defined in this document provides a road map for transforming Department of Education's Enterprise Architecture into an environment that supports the integration of disparate domains, platforms, data resources and architectural topologies. This architecture promotes reuse through industry standard component models while supporting the necessary qualities of service and enterprise strength scalability. The conceptual architecture defined in this document is comprehensive. Like its legacy counterpart, the time it will take to fully implement this architecture will be measured in years. The next step in implementing this architecture is a series of prototypes. These prototypes will be used as test beds to:

- Determine the best products to support the development of the architecture.
- Implement the necessary frameworks used by application development teams.
- Train the staff to develop applications using the selected products and component models.
- Change the culture at Department of Education to embrace component based development.

7 Acronyms

Table 19 – List of Acronyms

Acronym	Description
ACL	Access Control Lists
AE	Advanced Edition
AM	Application Interface Messaging
API	Application Program Interfaces
CA	Certificate Authority
CB	Component Broker
CICS	Customer Information Control System
CORBA	Common Object Request Broker Architecture
CRM	Customer Relationship Management
DDL	Database Definition Language
DMZ	Demilitarized Zone
DOE	Department of Education
DRE	Dynamic Reasoning Engine
EAI	Enterprise Architecture Integration
EDD	Enterprise Data Domain
EJB	Enterprise Java Bean
ETA	Enterprise Technical Architecture
ETL	Extract Transform Load
GCA	Gateway CommArea
GTP	Gateway Transfer Program
GUI	Graphical User Interface
HPUX	Hewlett-Packard UNIX
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Secure Hyper-Text Transport Protocol
IBM	International Business Machine

Acronym	Description
IHS	IBM HTTP Server
IIOP	RMI / Internet Inter-ORB Protocol
IP	Internet Protocol
IT	Information Technology
ITA	Integrated Technical Architecture
JNDI	Java Naming and Directory Interface
JSP	JavaServer Pages
LDAP	Lightweight Directory Access Protocol
LDD	Legacy Data Domain
MQ	Message Queuing
MQAA	MQSeries Application Adapter
MQSI	MQSeries Integrator
MS	Microsoft
MVC	Model-View-Controller
ODS	Operational Data Stores
OLAP	On-Line Analytical Processing
OLTP	On-Line Transaction Processing
OO	Object-Oriented
ORB	Object Request Broker
OS	Operating System
OTS	Object Transaction Service
PAC	Presentation-Abstraction-Control
PKI	Public Key Infrastructure
RMI	Remote Method Invocation
SFA	Student Financial Assistance
SNA	Systems Network Architecture
SQL	Structured Query Language
SSL	Secure Socket Layer
TCP	Transmission Control Protocol

Acronym	Description
TO	Task Order
UNIX	Universal Interactive Executive
URL	Uniform Resource Locator
VIC	Viador Information Center
WLM	Workload Management
WWW	World Wide Web